OBJECT-ORIENTED PROGRAMMING

November/December 1991

Volume 9.6

Programmer's Journal™

4.95 Canada $5.95

A Windows Class Library
page 16

Trouble in Object City
page 26

Almost-OOP Windowing
page 30

Stalking the Midnight Vampire
page 53

Revisiting ASM Libraries
page 44

0 74470 74130 5 11

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

If your idea of 38
is getting "out of mem
you don't need

This Manifest memory analysis shows 618K of memory available for DOS programs after running QEMM 6.0. That's enough to avoid the dreaded "out of memory" message.

Here's how you can get more memory. Network drivers, TSRs and other utilities that usually take memory away from your programs have been moved up into high memory, automatically—using QEMM's 'optimize' command.

Manifest, our memory management utility, clearly shows you how your memory is being used.

This example system is a Toshiba, but most 386SX, 386 and 486 IBM® PCs and PS/2s™, and most 100%-compatible 'clones' allow the high memory breakthrough.

QEMM 6.0 Stealth technology 'remaps' ROMs, freeing-up as much as an additional 115K to be used by your favorite programs.

Quarterdeck expanded memory manager—QEMM-386—became the best selling utility in the industry because it does a better job of managing memory.

QEMM solves memory problems smoothly and invisibly. It 'pools' memory so that it's available in whatever form your programs need—expanded or extended. You don't even need to know the difference. QEMM does it all for you. Instantly. All other managers require you to manually allocate memory and re-boot every time you need to change.

Our new version 6 is better still. First, we improved our unique 'optimize' feature. It's still as easy as ever to use: all you have to do is type 'optimize' and QEMM automatically seeks out TSRs and device drivers and moves them out of your PC's conventional memory and into high memory. Now it lets you set up "what-if" scenarios and 'play' with combinations of your own.

Either way, QEMM-386 v6 finds more memory for your programs than any other memory manager. Period.

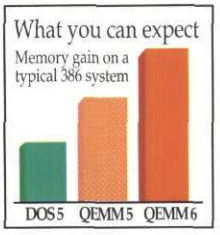
And QEMM monitors your memory—checking to be sure TSRs and utilities can be moved safely before it does so.

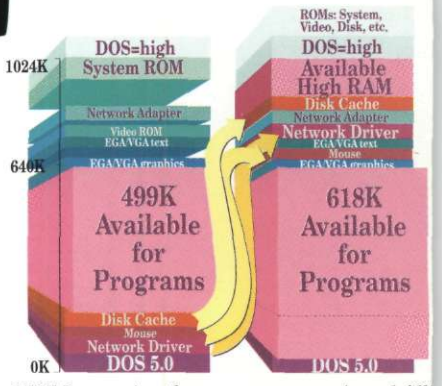
Who'd have thought there'd be another 115K of memory in your PC?

Breakthrough 'Stealth' technology results in a gain of up to 115K of high memory on many PCs by taking advantage of the memory map of most IBM, Compaq and 100% compatible PCs to 'map' ROM into other areas of memory. Only our memory wizards fully understand the technology behind it, but every user can appreciate the huge increase in available memory.

Of course, not every PC has 115K of extra high memory, but every PC can benefit from 'Squeeze'—our new feature to manage all those TSRs that need more memory at start up and less when they're resident. Their memory allocation is temporarily increased at start up, then squeezed down when it's no longer needed.

QEMM automatically uses idle VidRAM to produce a further 96K gain on EGA and VGA-equipped systems when running character-based programs.





QEMM reorganizes the way your memory is used, filli memory gaps and squeezing every last ounce of performance out of the software you now own.

©1991 Quarterdeck Office Systems. All trademarks or registered trademarks are property of their respective ow

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

'486 performance ory" messages faster, ur new QEMM.

QEMM gives you more memory with less pain.

ew breakthroughs aside, QEMM does s basic job better than any other emory manager. For example, the DOS 0 EMM uses a manual, trial-and-error rocess to optimize memory while EMM does it automatically.

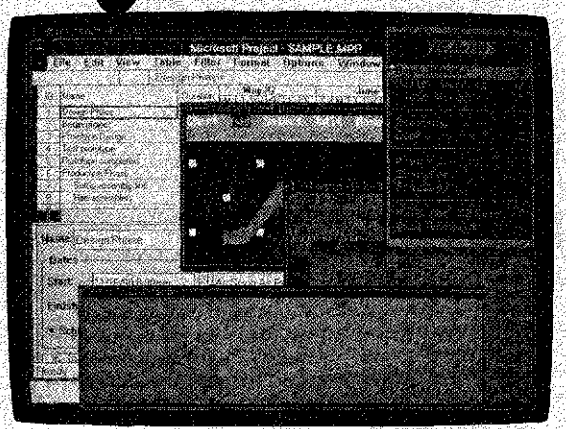
And QEMM 50/60 v6 offers these atures (except Stealth) and 'loads high' n IBM PS/2 Models 50 and 60. DOS 5.0 oesn't.

QEMM isn't just for power users. It's r anyone who wants to get the most out the PC. Whether you're using a PS/2 odel 50 and DOS 3.3, a 486 with 5.0, or mething in between, we can improve the ay it works.

rior versions of QEMM have won lots of awards—and become the #1 selling utility in the PC industry.

)on't leave DOS without us.

EMM provides an additional 8-24K of nventional memory to Windows 3 hanced mode—maximizing memory to lp make everything run better.



Our new DESQview-386 version 2.4 incorporates the latest memory-maximizing QEMM technology. That means it provides an even better solution for users who want low-overhead, high performance windowing and multitasking.

While DOS 5.0's EMM isn't even compatible with Windows' Standard Mode!

QEMM is the best memory manager to use with any multitasking software on the PC. Period.

Quarterdeck Manifest maps your way to the gold.

Our award-winning memory utility gives you complete and clear understanding of how your memory works.

You can actually see where programs, TSRs, network drivers and utilities run. Check memory speed. And find the best way to use all of what you've got.

Manifest is included right on the disk when you buy QEMM or DESQview-386.

New DESQview-386 A performance bonanza for demanding users.

DESQview-386 is our combination of QEMM-386 and DESQview. The result is a simple multitasking environment that works similarly to Windows 3.0, without all the memory and cpu 'overhead' associated with an added-on 'graphical interface.'

Our memory breakthroughs result in real benefits to users with lots of memory demands—especially network users. DESQview-386 lets you run programs in several windows side-by-side while using only about 9K of conventional memory on a typical PC.

Upgrade Policy

QEMM-386 v6 from any version of QEMM.....$30
QEMM 50/60 v6 from any previous version.....$30
QRAM 2.0 from any previous version.....$25
Manifest 1.1 from any previous version.....$20
DESQview 2.4 from any previous version.....$25
DESQview-386 v2.4 from any previous version $50

Priority Service customers: call for special pricing. Upgrade pricing good through 12/31/91.

Special Quattro Pro Bundle Offer

Upgrade from QEMM to DESQview 386 and get Manifest plus a set of DESQview Companions plus a free Special Edition Quattro Pro for.....$99

Act quickly—quantities limited.

As you might expect, its low memory demands mean DESQview-386 runs fast. So fast, you might not need to upgrade your computer to achieve a jump in productivity.

Quarterdeck productivity programs help you get more performance out of your PC investment today. And tomorrow.

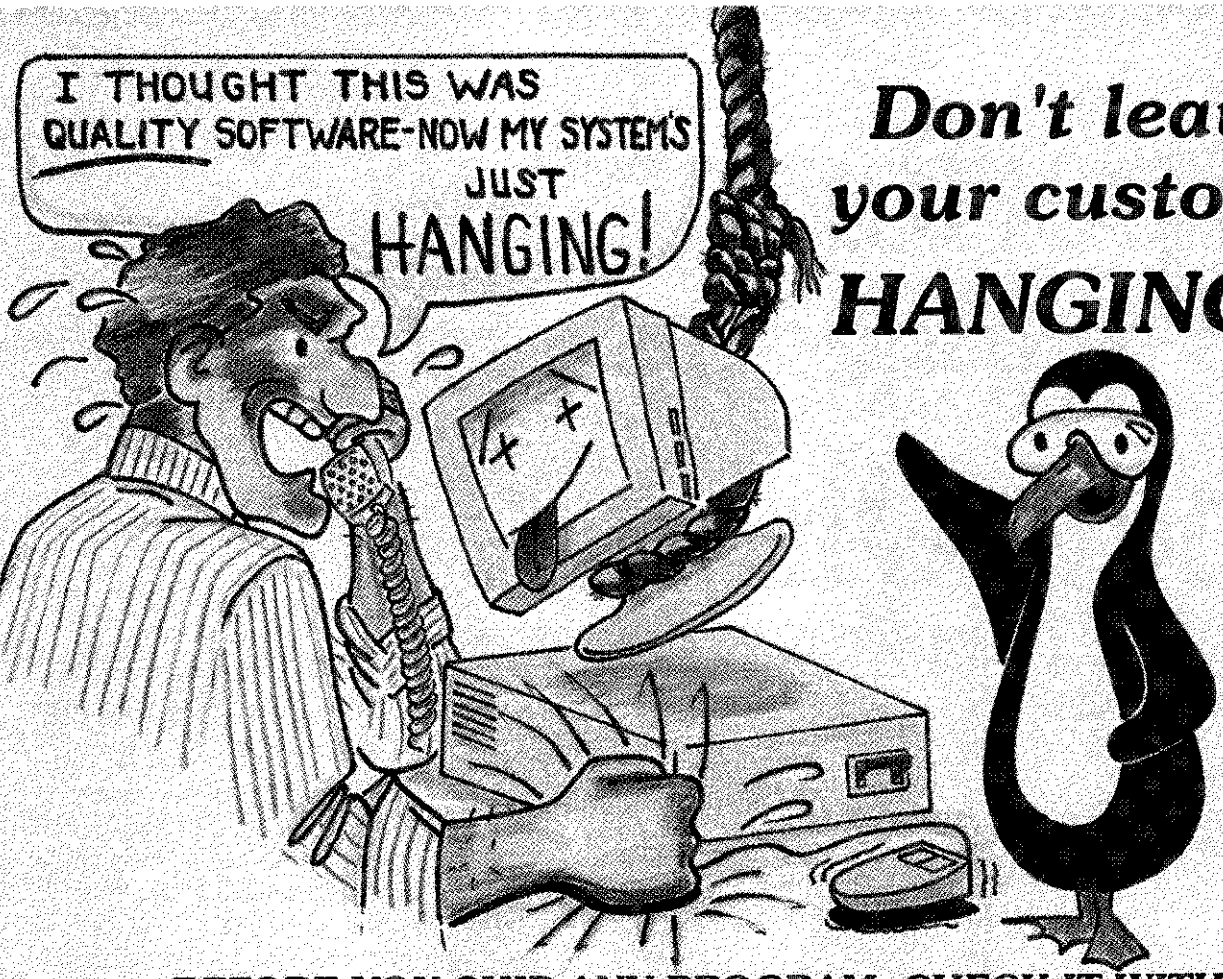
Quarterdeck

Quarterdeck Office Systems, 150 Pico Boulevard, Santa Monica, CA 90405 (213) 392-9851 Fax (213) 399-3802

For orders only, call toll-free (800) 354-3222 7AM-5PM PST.

Circle 1 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



Don't leave your customers HANGING. . .

BEFORE YOU SHIP ANY PROGRAM, CHECK IT WITH

COMPUTER LANGUAGE PRODUCT EXCELLENCE AWARD 1990 COLA

BOUNDS-CHECKER™

WINNER PC MAGAZINE BEST OF 1990

JANUARY 15, 1991 PAGE 168

FINDS OUT-OF-BOUNDS MEMORY ACCESSES *AUTOMATICALLY*

Your program may have 10,000 to a million lines of code. It may occasionally hang mysteriously or it may appear to run flawlessly every time. But under DOS, how can you ever be sure that your program is not corrupting memory it does not own? The only way to be 100% sure is to BOUNDS-CHECK before you ship.

To use BOUNDS-CHECKER you build your program with debugging information (we support most compilers including Microsoft, Borland & JPI). Then you just type <BC file-name>. BOUNDS-CHECKER sets up the 386™ /i486 for protection and lets your program fly. If your program accesses memory it does not own or overwrites its own code, BOUNDS-CHECKER pops up displaying the offending SOURCE-LINE or instruction.

Programming under DOS is a gamble, so why not stack the odds in your favor--CALL TODAY.

(603) 888-2386

OTHER FINE NU-MEGA PRODUCTS. . .

The ultimate systems debugger.

Debug:
- Interrupt routines
- Device drivers
- T&SRs
- ROMS
- Applications
- Overlays

Soft-ICE™

Features:
- Break out of a hung program
- Real time Break-Points
- Back-Trace history
- Works with other debuggers

1988 BYTE AWARD OF DISTINCTION

COMING SOON
Soft-ICE for Windows

Get inside Windows with the first REAL Windows system level debugger. $386

Run CODEVIEW for Windows on a single monitor
CV/1.....$129
• Runs in a window • No annoying flash •
• Runs on any display that supports windows •

Call by 4:00 PM EST TODAY and ask us to EXPRESS you an info packet. In most cases you will receive it by 10:30 AM tomorrow. (USA only)

All Nu-Mega products require a 386, 386SX or 486. MS-DOS and Codeview are trademarks of Microsoft Corp. 386 is a registered trademark of Intel Corp. Nu-Mega, BOUNDS-CHECKER, Soft-ICE and CV/1 are trademarks of Nu-Mega Technologies, Inc.

™ Nu-Mega TECHNOLOGIES INC

P.O. Box 7780 • Nashua, NH • 03060-7780 U.S.A.
(603) 888-2386 • Fax (603) 888-2465

BOUNDS-CHECKER $249
Soft-ICE $386

Special Offer
Buy BC & S-ICE ..... Save $100

30 Day
Money Back Guarantee

Circle 2 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Programmer's Journal™

November/December 1991 Volume 9.6

You won't find a bug in this OOP—just a tasty stew of objects, classes, and hierarchies in an assortment of flavors. This issue serves up a hearty, rib-sticking meal—so go on and play with your food all you want, and be sure to save room for dessert.

Cover Art: Digital Artworks

FEATURES

OBJECT-ORIENTED **16** **A TINY WINDOWS CLASS LIBRARY** *By John Dimm*
Simple is beautiful. John shows how to shrink an entire Windows program down to just eight lines using C++ and make the switch to object-oriented compilers and class libraries at your own pace.

OBJECT-ORIENTED **26** **TROUBLE IN OBJECT CITY** *By Brett Glass*
Can we rescue OOP from the strict hierarchies it's hanging around with? Brett sees the current object-oriented paradigm as perilously close to being cast into the stone of OOP languages.

C **30** **A WINDOW SYSTEM FOR TEXT EDITING** *By Allen Holub*
This is it—Allen winds up his layered windowing system. The final installment includes all the functions needed by an editing engine to implement a window-based editor. For additional elegance, the code is written in an object-oriented manner.

ASM **44** **LIBRARIES REVISITED** *By John Otken*
Nothing's better than a good library, especially if you're programming in assembly language. John uses the standard C library and other well-known models as templates for his ALIB library, which contains about 120 usable functions.

SYSTEMS **48** **SAVE YOUR VIDEO MODE** *By Walter Oney*
Are you tired of having your video mode trashed by applications that make rude and hasty exits? Don't get mad—get VIDSAVEd. Screen preservation and restoration are just a program away. Walt also shows you how to go about wrapping an EXE program with a COM file.

SYSTEMS **53** **THE MIDNIGHT VAMPIRE** *By Murray L. Lesser*
Just when you thought it was safe to go back to the keyboard, you find a day has been sucked from your DOS calendar. Murray has a program to foil the vampire—with GARLIC, naturally—and he urges readers to regard "the experts" with a healthy grain of salt.

GRAPHICS **59** **FAST HERCULES LINE DRAWING** *By Nicholas Wilt*
The venerable Hercules standard isn't losing any ground in the monochrome graphics arena. Going back to basics, Nicholas presents assembly language code for fast line drawing on banded graphics cards such as the Hercules monographics, the AT&T 6300, and the original CGA.

COLUMNS

PJ FORUM **9** **SOFTWARE TESTING TECHNIQUES** *By Warren Harrison*

REDIRECTIONS **13** **PINK NOISE** *By Frank D. Greco*

AT LARGE **67** **KEEP ON TRUKING** *By Dave Thompson*

CONSULTING **69** **C++ FOR BIG PROJECTS** *By M. Steven Baker*

ARMCHAIR **74** **THE WORD IS OOP** *By M. Steven Baker*

OFFRAMP **88** **HOW MUCH IS THAT MULTIPROCESSOR?** *By Hal W. Hardenbergh*

DEPARTMENTS

4 **EDITOR'S NOTE**

6 **LETTERS**

60 **BBS LISTINGS**

76 **AD INDEX**

80 **PRESSROOM**

86 **PJ BILLBOARD**

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

EDITOR'S NOTE

Just back from the forests and granite mountaintops of Maine, I'm able to ponder the sea of press releases on my desk with renewed energy. The hottest software prospects are Windows applications and object-oriented programming (OOP) products. The Fall Comdex bash and new-product parade is just around the corner as I write this, even though this month before Comdex seems a bit calm and quiet here.

According to the media mavens, any language can become object-oriented—with a few changes, C becomes C++, or Pascal becomes Object Pascal. The popularity of these languages over other OOP tongues (such as Smalltalk) is no doubt due to the optimism of those software developers who think that C and Pascal programmers can be easily made into OOP experts. Unfortunately, new ideas are hard to assimilate, and OOP is a whole new way of thinking about software development. Although Borland has successfully marketed C++ and Pascal with Objects, most users are still programming without using object technology.

A major stumbling block for OOP technology is its lack of standardized class libraries for basic functions. Using OOP for Windows programming presents a case in point. The Windows API is large, inconsistent, and expanding with each new release. (Given the undocumented Windows calls that many applications require—debuggers and profilers, for example—the API looks even uglier.) As a C programmer, I can't understand why the API designers didn't use structures for passing parameters to many Windows functions. A standard class library that encapsulates the Windows API into a smaller set of object-based functions and data structures would be incredibly useful. The operative word here is "standard"; changing compilers or development environments shouldn't mean that programmers need to learn yet another "standard" class library.

Are standards in the offing? A few years ago, Microsoft officially supported Glockenspeil C++ for Windows development, but don't expect the AFX libraries that will surface with C/C++ 7.0 to have any connection to previous Windows classes. The closest thing to standard classes comes from Borland and Whitewater (developer of Actor), primarily because the Whitewater folks developed the Borland Pascal and C++ enhancements.

DOS LIVES THROUGH THE OS UNCERTAINTY

The 386 development-tools market entered a new phase when Intel started bundling a royalty-free DOS extender with its C Code Builder Kit. By year's end, every 386 compiler vendor will be supplying similar features. Rational Systems, although new to the 32-bit world, will win big with its DOS/4G product. Priced to sell, DOS/4G will be bundled with Watcom, Microway, and MetaWare compilers. Phar Lap has added support for Borland C++ and Microsoft Fortran in the new release of its 286 DOS extender, which emulates just enough of the Windows API to allow full use of Borland's compiler, linker, and Turbo Debugger for Windows.

Windows 3.1 will shed its real-mode beginnings as it matures. However, Microsoft has befuddled the DOS GUI marketplace with its premature talk of Windows NT for the 386. In the interim, Watcom and MetaWare both ship fine products that allow 32-bit Windows applications to be delivered now.

The PR battle between Microsoft's Windows and IBM's OS/2 2.0 is gearing up for the end-of-year blitz. Can "sleeping giant" IBM ride out the Windows wave? My own experience with IBM makes me none too hopeful. It took four weeks and numerous phone calls to get a current beta of OS/2 2.0, and I'm still waiting for the IBM SDK to replace the Microsoft-supplied tools released in March. If my experience is typical, very few OS/2 ancillary development tools are likely to be emerging soon. Compare this with the deluge of modest Windows tools and products emanating from smaller companies; the sheer mass of this software adds credibility and substance to Windows as the GUI and OS for the PC world. IBM may respond to requests from its Fortune 500 clients, but it has abandoned the millions of AT clones in the mainstream DOS market. Can IBM deliver a "better Windows than Windows" by the time the snow hits the slopes? My crystal ball says that its chances for success are unlikely indeed.

Steven Baker

Programmer's Journal

Editor
M. Steven Baker
Editorial Director
J.D. Hildebrand
Managing Editor
Cecelia Hagen
Associate Editor
M. Elizabeth Hardinger
Contributing Editors
Brett Glass, Frank D. Greco, Hal W. Hardenbergh, John Otken, Tom Swan
Founder
Greg Estes

Publisher
Elizabeth Oakley
Advertising Manager
Jay Moore
Account Executive
Maryann Kearns
Advertising Coordinator
Bonnie Chin
Circulation Manager
Randy Randolph
Circulation Promotion
Julia Moore
Fulfillment Manager
Pamela Coleman

Production & Art Director
Wanda Walker
Production Assistants
Margie Hanna, Nicole Harris

Financial Controller: Thomas Graham
Office Staff: Tina Bates, Jill Edwards, Bruce Searl

Business and Editorial Offices: *Programmer's Journal,* P.O. Box 70167, Eugene, OR 97401-0110, (503) 747-0800, (800) 234-0386, Fax: (503) 746-0071, BBS: see page 60

Subscriptions: 1 year (6 issues) $29.70

Articles and letters in *Programmer's Journal* represent the views of the authors and not necessarily those of the publisher. Letters to the editor become the property of *Programmer's Journal* and may be edited for publication.

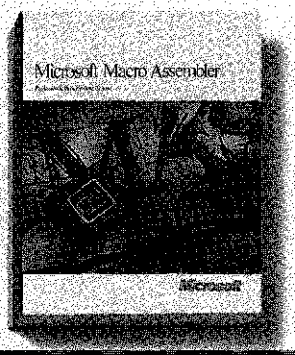
Note that all code and listings in *Programmer's Journal* appear "as is," and *Programmer's Journal* makes no guarantee or warranty of their quality, performance, accuracy, or suitability for any particular purpose. *Programmer's Journal* assumes no responsibility or liability to any person or entity with respect to any damage caused or alleged to be caused by their use.

Advertising: For ad rates, deadlines, and placement, call Jay Moore or Maryann Kearns at 800-234-0386, or write to *Programmer's Journal,* P.O. Box 70167, Eugene, OR 97401-0110. *Programmer's Journal* is produced using the software products Ventura Publisher® and CorelDraw. The PostScript® files are output on a Linotronic® 200.

Programmer's Journal (ISSN 0747-5861) is published bimonthly by Oakley Publishing Company, 150 N. 4th Street, Springfield, OR 97477-5454. Second-class postage paid at Springfield, OR, and additional mailing offices. USPS # 000 293. **POSTMASTER:** Send address changes to *Programmer's Journal,* P.O. Box 70387, Eugene, OR 97401-0121.

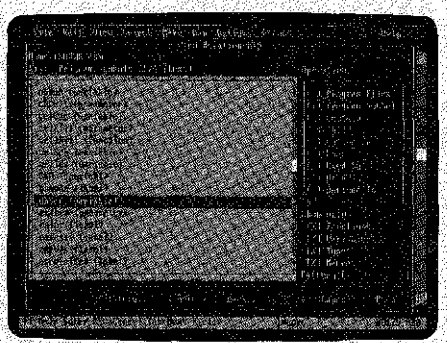
© 1991 *OAKLEY PUBLISHING COMPANY*

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

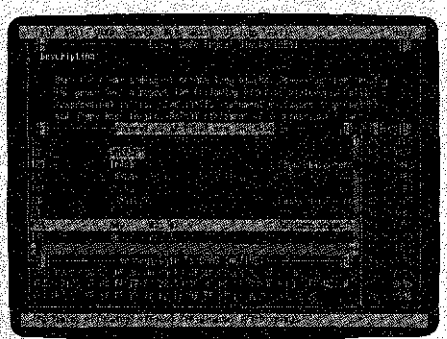


MICROSOFT SOFTWARE DEVELOPMENT TOOLS

Microsoft MASM 6.0. Now 23% less painful.



Source Browser lets you explore the relationships between functions, variables, types, and macros in MASM-only and mixed-language programs.



Microsoft CodeView gives you instant access to all of your source and data, including memory, registers, and flags. You can also access Microsoft Professional Advisor online documentation to get instantaneous assistance.

We know exactly how you feel about assembler: Low-level language, high-level frustration.

But the new Microsoft® MASM 6.0 relieves the stress and strain of Assembler programming (exactly how much depends on how crazed it made you in the first place).

MASM 6.0 has additional high-level language extensions for easier programming. It has the ability to convert C header files into MASM include files, so you don't have to. And the intelligence to optimize the shorter jumps and eliminate NOPs and phase errors from the longer ones.

You also get 386™/486™ instruction set support.

And a Programmer's WorkBench that supports development of MASM-only or mixed-language programs. All under one seamlessly integrated environment.

So call (800) 541-1261, Dept. Q59, for a free white paper.

And fast relief from headache pain.

For more information inside the 50 United States, call (800) 541-1261, Dept. Q59. Customers in Canada, call (416) 568-3503. Outside the U.S. and Canada, call (206) 936-8661. *© 1991 Microsoft Corporation. All rights reserved. Microsoft, the Microsoft logo and CodeView are registered trademarks and* Making it all make sense *is a trademark of Microsoft Corporation. 386 and 486 are trademarks of Intel Corporation.*

New Features

- High-level language data types and control structures make mixed language programming easier. Including: PROTO/INVOKE, TYPEDEF, .IF/.THEN/.ELSE, and .REPEAT/.WHILE.
- Macro language extensions such as: @SubStr, @CatStr, @Date, & @Time.
- H2INC converts C header files to Assembler include files.
- Automated Jump shortening speeds execution.
- 486 and expanded 386 instruction set support with flat model (32-bit) support.
- *n*-pass Assembler averts phase errors and Assembler-generated NOPs.
- User-defined Prologue/Epilogue customizeable for Windows.
- /Zm switch plus other features to help accommodate your MASM 5.1 code.
- Significantly improved Assembler speed.
- Programmer's WorkBench v 1.1 and CodeView® v 3.12 make it easier to develop stand-alone .asm and mixed-language programs.
- Source Browser gives instant access to data declarations and function definitions.

PROGRAMMER'S TIPS

The new INVOKE statement automates procedure calling, by handling all pushing and popping of parameters on the stack. Example:
INVOKE printf, ADDR fmt1st,
ADDR str2, dat1, dat2

If you want MASM 5.1 compatibility, start with the MASM.EXE compatibility driver or use the /Zm switch with ML.EXE.

Microsoft
Making it all make sense

Contact this advertiser directly

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LETTERS

ODESSA ON MY MIND

I was working on a database engine for a statistical analysis system when I received my copy of *PJ* 8.6. The "B-tree indexing" article by M. Steven Baker at once seized my attention. The article is an invaluable source on index file formats. It interests me from a theoretical point of view and as a guide to index file formats of most popular database software.

I used the information in Baker's article on FoxBASE and FoxPro IDX file structures in my project. But even after repeated attempts, it refused to work properly. I spent a Sunday dumping FoxBASE index files and converting decimals to hexadecimals and vice versa. This was a frustrating activity, and I saw Norton's DiskEdit screens in my sleep.

I believe I found three inexactitudes in Baker's article. First, the key is going first in the key group, and the block offset or record number follows it. Both the block offset and the record number are long integers in Motorola format. In the index header, the second long integer is an offset of the end of file, and the third is a pointer to the chain of free blocks.

Enclosed are my changes to Listing 3 from Baker's article (see Listing 1).

I hope my additions will help those readers who appreciate Baker's article but who can't use the index file structures as he describes them.

Oleg P. Pilipenko
Odessa, USSR

THE EDITOR RESPONDS: I also spent many nights seeing index blocks in my sleep, so thanks for the help.

ILLUSTRATION BY JIM CARPENTER

INFORMATION, PLEASE

I enjoyed the "DOS extenders" article by M. Steven Baker (*PJ* 9.5). It was informative and very useful for what I'm presently working on. But it had two problems.

Mr. Baker stated that Microsoft publishes a five-page technical note that describes Windows 3–supported DOS calls and DPMI functions. However, when I called the Microsoft Product Information's 800 number to request this publication, they knew nothing about it. It would also have been helpful to know the DPMI specification version that this technical note covers.

I suggest that in future articles, you give complete references so the reader can follow up on the information.

David A. Steininger
Santa Clara, CA

THE EDITOR RESPONDS: Documentation on the DOS calls supported by Windows 3 is available in Microsoft's "Windows Developer's Notes" (800-426-9400). This set of notes ($20, plus tax, shipping, and handling) includes source for a sample Windows setup program. Microsoft will also provide a "Windows Resource Kit" ($15) that documents setting up Windows for various networks and custom applications. The kit includes a general overview of how Windows, DPMI, and VCPI fit together.

Windows 3 supports most functions in the DPMI 0.9 specification, available for free from Intel (800-548-4725). Version 3.1 of Windows is to support the same DPMI 0.9 spec with one function added from DPMI 1.0 for math coprocessor emulation.

BACK TO BASIC LIBRARIES

Thanks to Murray Lesser for his "BASIC libraries" article (*PJ* 9.3). I have found the utilities useful.

I do, however, have two comments. First, I don't see the value of linking with stub files in this case. Because the utilities are designed to be used in the BASIC development environment, where the runtime libraries are available, I linked them with those runtime libraries and wound up with executables of 5.6K for PBBI and 9.9K for DOLIB.

My second comment is a suggestion. I do a lot of development in the ZBX environment and noticed that the addition of five lines of code to DOLIB will let it update a personal QuickLibrary as well (Listing 2). I added the command-line switch to make the QuickLibrary optional, because it needs to be rebuilt each time and therefore slows the process down.

James Brodsky
Shell Beach, CA

LISTING 1

```
/*   Code to accompany letter by Oleg Pilipenko   */
/***    FoxBASE and FoxPro IDX file structures   ***/
#define BLOCK_SIZE 512   /* our B-tree block size   */
#define KEY_LENGTH  30   /* our example search-key expression length  */
#define MAX_KEYS    13   /* This number varies based on key length. The
                         /*   following example is based on a 30-character
                         /*   key expression. */

typedef struct index_header {
    long    root_offset;   /* offset in file of the root block    */
    long    eof_offset;    /* offset in the file
                           /*   where new blocks are added */
    long    free_ptr;      /* pointer to the chain of free blocks */
    short   key_len;       /* length of the search key expression */
    short   unique;        /* flag if index on unique keys only */
    char    expression[100]; /* index expression used for the search key */
} idx_header;

typedef struct key_group { /*FoxBASE IDX key entry  */
    char    searchkey[KEY_LENGTH]; /* our search key   */
    union {
        XLONG block_offset; /* offset in file for root & branch nodes   */
        XLONG rec_num;      /* in leaf nodes, record number   */
    }
} idx_key_group;
```

LISTING 2

```
'-- Code to accompany letter by James Brodsky----------------------:
  Text$ = COMMAND$
'new code:
  IF RIGHT$(Text$, 2) = "/Q" THEN
    Text$ = RTRIM$(LEFT$Text$,LEN(Text$) - 2))
'continue existing code:
  'Make sure there is a <modname> to add:
        :
        :
  PRINT #1, "LIB PBF-+"; ModName$; ",PBF.IDX;"
  PRINT #1, "IF ERRORLEVEL 1 GOTO END"
'new code:
  IF INSTR(COMMAND$, "/Q") THEN
    PRINT #1, "LINK /Q PBF.LIB,PBF.QLB,,QBXQLB.LIB;"
    PRINT #1, "IF ERRORLEVEL 1 GOTO END"
  END IF
'continue existing code:
  PRINT #1, "DEL "; ModName$ ".OBJ"
        :
        :
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

WE HAVE AN OBJECTIVE POINT OF VIEW

...with the **Turbo Vision Development Toolkit** for Turbo Pascal 6.0

We view traditional programming that requires recompilation to see changes to the program interface in a completely new and different way! **Turbo Vision Development Toolkit** lets you interactively create or change dialog boxes, menus and string lists in the Turbo Vision application framework. You can use the **Resource Editor** to develop and see Turbo Vision interface objects exactly as they will appear in your final application. The Resource Editor saves the objects to a resource file that can be loaded into your application with a single method call.

The Toolkit also includes a utility to translate your resource files into Windows 3.0 resource script files. This makes it easier to port your Turbo Vision applications to Windows.

An ongoing objective...

We couldn't and didn't stop with just powerful utilities. We've built a number of objects that make your development easier and your applications more bullet-proof. Just some of these objects are an **Enhanced Help Facility** to display context-specific help information; a **Memory Monitor** to record and display memory usage; an **Event Monitor** to display all or selected events processed by Turbo Vision.

These objects are especially useful during program development and debugging, and we've made it easy for the objects to be added to or removed from your application with a single statement.

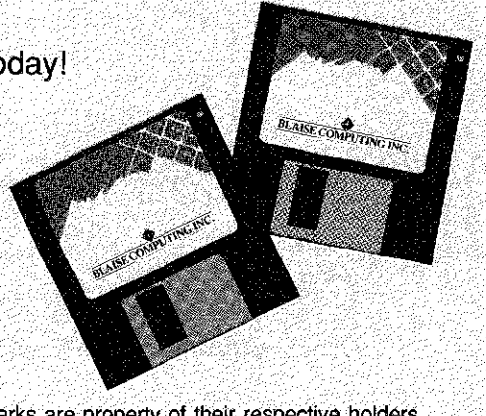
And, there's more...

Of course, the **Turbo Vision Development Toolkit** comes with a comprehensive **Reference Manual** and **Tutorial**. The source code for all objects and utilities is included.

Consider our point of view...

The **Turbo Vision Development Toolkit** requires Turbo Pascal 6.0 and costs just $149. We're so convinced that you'll find these utilities and objects essential that if during the first 60 days you're not completely satisfied, we'll refund your money. That's a 60-day money back guarantee!

Call our order department **toll free (800) 333-8087**, today!



BLAISE COMPUTING INC.

819 Bancroft Way Berkeley, California 94710 (510) 540-5441

Circle 3 on Reader Service Card

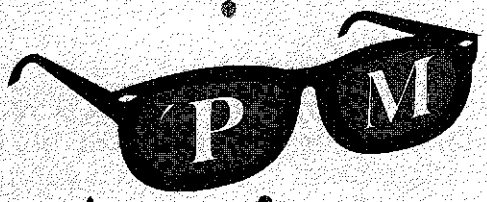
Trademarks are property of their respective holders.

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Conquer OS/2 !!

Now OS/2 can respond to your every whim

Introducing:



Assistant

PM Assistant is the **first** and **only** Automation and Customization Utility for OS/2 Presentation Manager

- Automate repetitive tasks with **mouse and keyboard input macros.**
- Play keystrokes to any PM application or full-screen session.
- Start, position, and initialize your PM applications **automatically.**
- Switch to and position PM applications and full-screen sessions with user-defined **hot-keys.**
- Execute any macro from within any PM application or full-screen session with user-defined **hot-keys.**

- Set, view, and change *any* program's **OS/2 process priority.**
- Assign **accelerator keys** and rearrange the menu commands of *any* PM application.
- Speed dialog box interaction with Speed Button.
- Prevent damage to your monitor with **Screen Saver.**
- Protect your computer from unauthorized access with **Password Protection.**
- Create turn-key procedures or demonstrations of your PM programs with the **demonstration utility.**

Now, all your favorite tools in Dos and Windows are available for Presentation Manager along with features unique to OS/2. PM Assistant comes with on-line help, an unconditional satisfaction guarantee, and a **free program** - DiskWatch, a disk free-space monitor. PM Assistant runs on OS/2 versions 1.10 or greater, and is now available at a special introductory price:

PM Assistant.....$125.00

Take control of OS/2, order today!

1-800-745-7757

Utilis Advanced Tools Now

Trademarks are property of their respective holders.

P.O. Box 15905
Seattle, WA 98115
(206) 467-4025

Circle 4 on Reader Service Card

COMPUTER LANGUAGE PRODUCTIVITY AWARD 1990

INSTALL.BAT

First impressions last.

INSTALL 3.0 gives your software product a professional introduction.

Installations that rely on batch files look amateurish and have limited error handling. INSTALL 3.0 creates an elegant installation procedure that will increase user's confidence in your software product.

INSTALL 3.0 comes with many samples you can modify and use immediately; no programming usually needed. All available RAM is used for fast file transfers.

- Features very sophisticated error-handling
- Does high performance data compression
- Can safely and reliably modify system files
- Allows selective installation of parts of your program
- Complete source code included
- No royalty charges
- Free technical support

INSTALL 3.0 detects monitors, DOS versions, coprocessors, etc. to create machine-specific installations without asking questions.

INSTALL 3.0 has been used for over four years by some of the biggest (and smallest!) names in the industry, in the U.S. and abroad, to install millions of copies of programs. Make sure your software product makes a good first impression with INSTALL 3.0.

KNOWLEDGE DYNAMICS CORP.
P.O. Box 1558
Canyon Lake, TX 78130-1558
MasterCard/VISA/COD/POs welcome

$249.95 + shipping
Money back guarantee
Sales 800-331-2783 Ext 0196
Internat'l 512-964-3994
24 hr FAX 512-964-3958
24 hr BBS 512-964-3929

Circle 5 on Reader Service Card

The Programmer's Corner

— A Source Code Repository —

If you are reading this magazine, then you appreciate the art of programming. One of the most important tools for a software craftsman is example code. To meet this need, we provide **more than 9000 files** (over one gigabyte) of shareware and public domain source code and programs. We cover all popular languages including C, Pascal, and Basic. TPC even has hard to find source code such as OS/2 PM C source, C++ source, and Turbo Pascal 5.5 OOP source. Source code from this issue of *Programmer's Journal* is also available. New code is added daily, all in the interest of helping the professional programmer.

So call today: (301) 596-1180
(301) 995-3744
(301) 621-4636

300-9600 Baud
24 Hours-A-Day

Only $50 for a one year subscription

— TPC uses FastComm 9600 V.32 Modems —

TPC

The Programmer's Corner
5430 F Lynx Lane, Suite 224
Columbia, MD 21044

Circle 6 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Software testing techniques

By Warren Harrison

Testing software is one of the most tedious and unwelcome tasks programmers face. Not only is testing way down there on the list of fun things to do, but it also usually comes late in the project. So even if you were looking forward to testing your software, you probably don't have time to do much of it because of the pressure to get the product out the door.

Because of these time constraints, much of the academic research in software engineering in general—and software testing in particular—seldom gets put to use. No matter how many statistics we wave about or how many logical arguments we construct, it's still hard to argue with the "viability equation": cash flow in > cash flow out. Companies or individuals that violate the equation for too long don't stay in business.

Even if you have adequate resources available for software testing, time to market may be an issue. Most software products must be introduced within a market window (I call this the Comdex Syndrome). Whether the market window is defined by an important trade show or by the introduction of a competing product, products released within the optimum time frame often can gain a significant competitive advantage.

As one of the final steps in software development, testing is often the first activity to suffer cuts when other parts of the process, such as design or coding, overrun their schedules or budgets. Nevertheless, you almost always have *some* time for testing. The challenge is to make *effective* use of that time.

What is meant by *effective?* How do you know whether your testing time is applied effectively? How do you know whether you're making more effective use of your testing time than the guy down the hall? There are many ways to define and measure testing effectiveness. One approach is to give each module of the software the same distribution of testing time per error. For example, suppose you have three modules: A, B, and C. A and B have 5 bugs apiece, and C has 50 bugs. An effective allocation of 60 hours of testing time would be one in which A and B each receives 5 hours of testing, and C receives 50 hours. You can quantify the effectiveness by computing the standard deviation of the average amount of testing time per bug found for each module. Thus, a perfect allocation of testing effort would yield a standard deviation of zero—that is, the time spent testing each module would be proportional to its number of bugs.

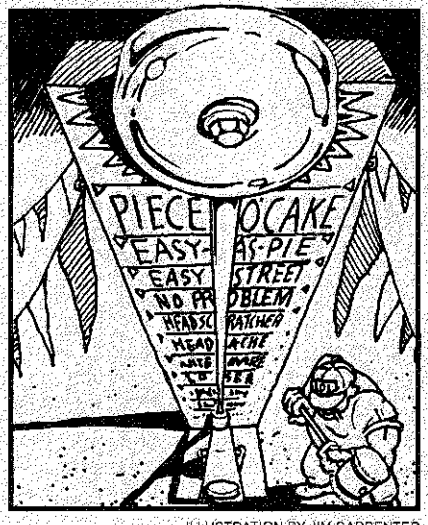


ILLUSTRATION BY JIM CARPENTER

TESTING ONE, TWO, THREE

Computing this allocation is easy if you know how many errors each module has, but it's a challenge to arrive at the allocation before you've started testing. However, there are techniques that can help you make your testing more effective.

You can allocate your time among parts of the program based on how much each contributes to the product's overall functionality. This method can make you feel more confident that the "important" parts of the product have been tested, but it falls short of a truly effective allocation of testing time, because the probability of a piece of code having an error isn't related to the importance of its function in relation to the rest of the product. If anything, it is likely to be inversely correlated: you're probably more careful designing and coding the "important" parts.

Another approach is to assess the degree of difficulty of the function being performed by each part of the code. For example, in the case of a compiler, you might consider code generation to be more difficult than parsing, and parsing to be more difficult than lexical analysis. However, unless you take into account the code that implements the various parts, you might make the wrong decision. Everyone knows that a particular function can be implemented in a number of ways, some more difficult than others. If you consider only the function and not its implementation, you have no way of knowing whether you've chosen the easy solution or the difficult one.

The most reasonable technique is to allocate your time based on the complexity of the code in each part of the program. It seems obvious that you're more likely to make mistakes within complicated code, and if the function performed by a particular module is considerably more complicated than the other functions in the program, that difference should be reflected in its code. The question is, how do you assess the complexity of a piece of code?

THE METRICS SYSTEM

Over the past decade, researchers have proposed a variety of *software complexity metrics* that attempt to assign a rating to a program based on how complicated the software is to understand, modify, or otherwise work with. Software complexity metrics are often classified according to the type of characteristics they consider. Perhaps the two most common classifications are *token-based metrics* and *control-flow metrics*.

Warren Harrison is director of the Center for Software Quality Research and an Associate Professor of Computer Science at Portland State University in Portland, Oregon. He is also chief scientist and vice-president of Research & Development at SET Laboratories, Inc., the North American editor for the Software Quality Journal, *and a member of the editorial board of* IEEE Software *magazine.*

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Token-based metrics view certain units of text within the program as *tokens*. A number is assigned to a program based on a summary of the occurrence of these tokens. Undoubtedly, the most popular token-based metric is lines of code, in which a line of text is considered a token. The token is often weighted to account for certain aspects of the product or environment. For example, equivalent assembler source lines (EASL)—which is becoming popular in Japan as an industry-wide measurement technique—adjusts lines of high-level code by appropriate weights so that systems written in different languages can be compared.

Another approach to token-based metrics, *software science*, distinguishes two classes of tokens: *operators* and *operands*. The tokens are summarized using four parameters: *unique* operators (η_1), *unique* operands (η_2), *total* operators (N_1), and *total* operands (N_2). These parameters are then synthesized into a variety of metrics. In particular, two metrics we will be interested in are *volume* and *effort*:

$$Volume = (N_1 + N_2) \log_2 (\eta_1 + \eta_2)$$
$$Level\hat{} = (2/\eta_1) \times (\eta_2/N_2)$$
$$Effort = Volume / Level\hat{}$$

A weakness with token-based metrics is that they usually cannot assess other aspects of a program, such as the flow of execution. *Control-flow metrics* can help remedy this problem. Perhaps the best-known of these, *cyclomatic complexity* (sometimes referred to as *VG*), summarizes the control flow of a program from a graphical representation of the program's flow of control (called a *flowgraph*). The measure can also be computed directly from the text of the source program by counting the number of decision points within the code (**if, while, for,** etc.) and adding 1 (interestingly, this transforms the measure into a token-based metric).

These three metrics—lines of code, software science, and cyclomatic complexity—are the most popular among users of software metrics, although many other metrics exist. The computation of each of these measures is illustrated in Figure 1. The question is, how useful is each of these in helping to allocate testing time?

A TEST OF THE TESTS

To answer this question, we performed a short experiment with some information we had about a moderate-sized (slightly more than 41,000 lines of C code) compiler implementation project. We had collected metrics and error history for each of the system's 20 logical modules. Table 1 describes the characteristics of the modules as well as the number of errors encountered during testing for each module.

Because the project had already been completed, we did not have an opportunity to use metrics in scheduling testing. We had

MAXIMUM POWER • MAXIMUM SPEED

Finally, A Linker That Compresses Overlays!

NOW ONLY $350

Only OPTLINK/Compress can compress overlays, yielding smaller, faster EXEs. And OPTLINK/Compress **links** faster than **LINK**. In fact, it's 4 to 20 times faster! And it comes with great new features you won't find anywhere else. So why waste any more time? Try OPTLINK/Compress – from the developer of OPTLINK/Windows and OPTLINK/2. For more information or a FREE 30-Day Evaluation, call us.
Dealer inquiries welcome. (412)282-0864

SLR SYSTEMS, INC.
1622 N. Main Street
Butler, PA 16001-1599
FAX (412) 282-7965
BBS (412) 282-2799

30-DAY MONEY BACK GUARANTEE

GREAT FEATURES

- MAXIMUM OVERLAYS – 8,192
- SUPPORTS FUNCTION POINTERS
- COMPRESSION OF ROOT AND OVERLAYS
- SMALL AND LARGE-MODEL OVERLAYS
- OVERLAYS TO SEPARATE FILE
- EMS/XMS CACHING
- SETS NO LIMITS ON THE NUMBER OF MODULES, LIBRARIES, PUBLICS OR SEGMENTS BEING LINKED

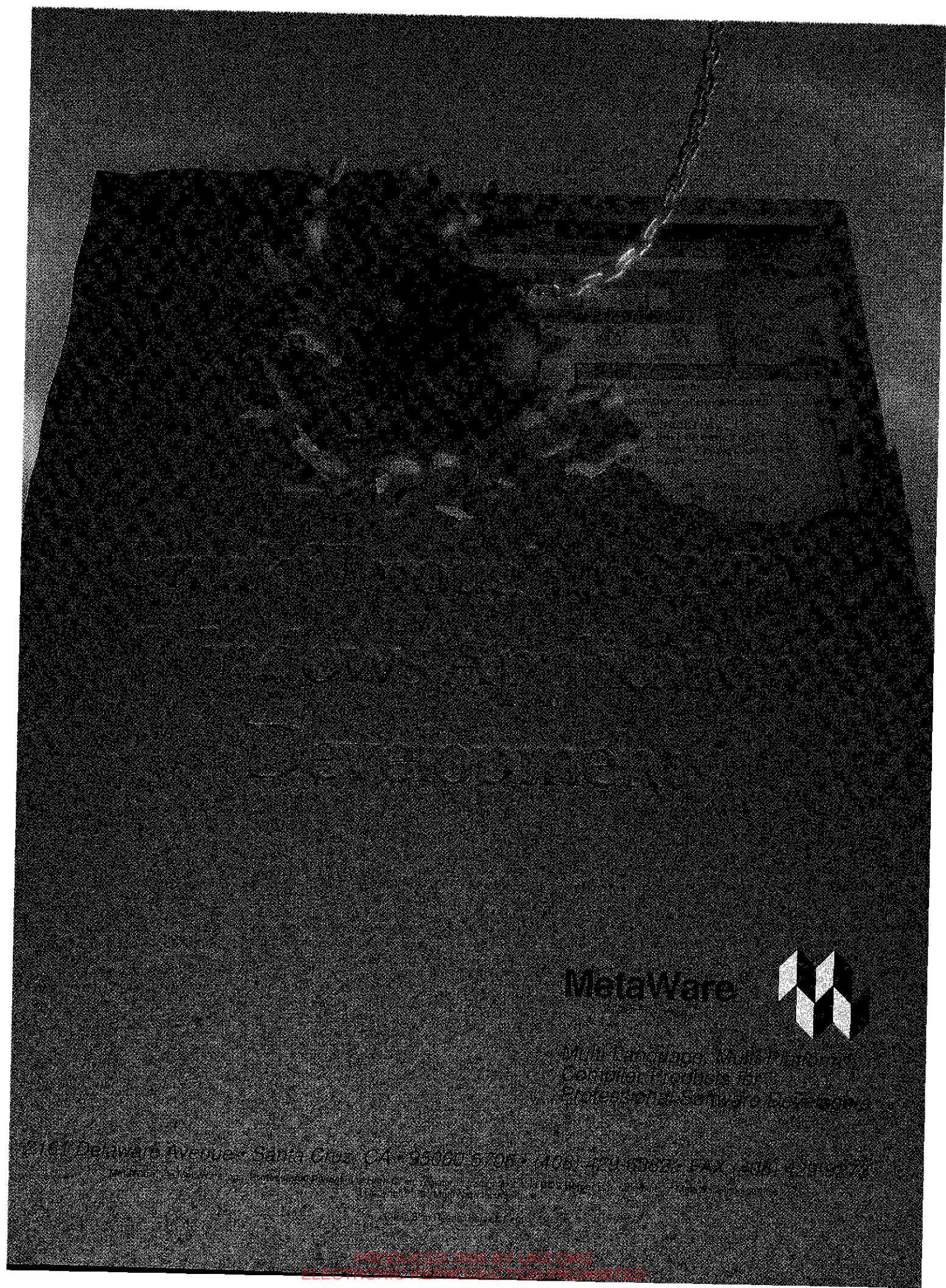
OPTLINK / Compress

OPTLINK/Compress, OPTLINK/Windows and OPTLINK/2 are trademarks of SLR Systems, Inc.

MAXIMUM POWER • MAXIMUM SPEED • MAXIMUM PERFORMANCE

Circle 7 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



MetaWare

Multi-Language, Multi-Platform
Compiler Products for
Professional Software Developers

2161 Delaware Avenue • Santa Cruz, CA • 95060 • (408) 429-6362 • FAX (408) 429-9273

PRODUCED 2006 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FIGURE 1: Example of metric computations

Sample Code Listing:

```
function SearchList(Token:StringType;List:ListType;
                    Count:integer):boolean;
var  Hi,Lo,Mid,Prev:integer;
     NotFound:boolean;
begin
   Hi:=Count;
   Lo:=1;
   Mid:=maxint;Prev:=0;
   NotFound:=true;
   while(PrevMid)and(NotFound)do begin
      Prev:=Mid;
      Mid:=(Hi+Lo) div 2;
      if List[Mid]then Lo:=Mid
      else if List[Mid]Token then Hi:=Mid
      else (* List[Mid]=Token *) NotFound:=false;
   end;
   SearchList:= not NotFound;
end;  (* SearchList *)
```

Metric Computation of Code:

```
η1 = 15          Volume = 363
η2 = 15          Effort = 5,990
N1 = 41          Cyclomatic Complexity = 4
N2 = 33          LOC = 18
N  = 74
```

TABLE 1: Module characteristics

Module	Lines	Effort	VG	Errors
A	1,491	3,315,443	126	6
B	3,459	16,410,628	208	32
C	3,724	18,746,624	292	22
D	504	386,167	14	1
E	1,158	2,794,050	89	10
F	1,127	2,479,381	64	6
G	3,625	18,728,417	320	17
H	3,531	65,295	14	12
I	3,149	501,413	38	20
J	1,569	5,330	4	12
K	337	110,596	22	1
L	2,452	4,256,003	128	3
M	1,816	4,634,226	115	4
N	718	2,178,060	75	3
O	1,371	724,419	33	14
P	4,611	31,581,032	255	26
Q	3,146	16,737,746	356	38
R	1,112	3,475,167	82	8
S	1,253	1,838,235	70	3
T	1,081	3,684,379	84	11

all the information we needed, however, so we were able to apply the approach after the fact to see how our testing effort allocation would have worked out.

We studied four different approaches to the allocation. The first approach, the *uniform allocation technique*, simply allocated the same testing effort to each of the 20 modules. If we've scheduled 100 hours to test the product, then each module would get 5 hours for testing. Obviously, the major drawback is that some modules had a far greater number of errors than other modules, yet each received the same amount of testing.

LOC-based allocation, the second approach, allocated testing effort in proportion to the number of lines of code in each module. With this technique, module P would get more than 11 hours, and module K less than an hour of testing effort.

We did a similar computation for each module using the software science effort and the cyclomatic complexity metrics. We referred to these techniques as *effort-* and *cyclomatic complexity–based allocation* techniques, respectively.

Because the goal is a uniform allocation of testing effort (that is, the number of hours spent per bug is the same across all modules), a useful way to evaluate each approach is to compare the resulting standard deviations for average number of hours per bug. This method rests on two assumptions:

- The number of errors encountered when testing each module is proportional to the actual number of errors in each module.
- More time spent on testing implies better testing.

The uniform allocation had a surprisingly large standard deviation of 1.39 hours. The lines of code and cyclomatic complexity allocations ended up with standard deviations of 0.43 hours and 0.45 hours, respectively. The results from the software science effort allocation were the best, with a standard deviation of 0.32 hours.

This study examined the usefulness of metrics in allocating testing resources, but the benefits of software metrics are not limited to testing: they have been applied successfully in a variety of other software development activities, from code reviews to maintenance. Metrics will become a more integral part of the programming landscape over the next decade. For those readers interested in more information about software metrics, the annotated bibliography should provide a good beginning. PJ

BIBLIOGRAPHY

- Curtis, B., et al. (1979). "Measuring the Psychological Complexity of Software Maintenance Tasks with the Halstead and McCabe Metrics." *IEEE Transactions on Software Engineering* **SE5**, pp. 96–104. Describes a controlled experiment in which performance on various maintenance tasks was related to software complexity measures.
- Fitzsimmons, A., and Love, T. (1978). "A Review and Evaluation of Software Science." *ACM Computing Surveys* **10**(1), pp. 3–18. A classic paper discussing the foundations and empirical evidence that support software science.
- Halstead, M. (1972). "Natural Laws Controlling Algorithm Structure?" *ACM SIGPLAN Notices* **7**(2), pp. 19–26. The seminal paper on software science—interesting reading.
- Halstead, M. (1977). *Elements of Software Science*. New York: Elsevier North Holland. The most comprehensive description of software science available—a must for anyone who is seriously interested in the area. Unfortunately, the book is out of print.
- Harrison, W., et al. (1982). "Applying Software Metrics to Program Maintenance." *IEEE Computer* **15**(9), pp. 65–79. Description and discussion of many popular complexity metrics in each of the four categories: control flow, size, data, and hybrid.
- Harrison, W. (1988). "Using Software Metrics to Allocate Testing Resources." *The Journal of Management Information Systems* **4**(4), pp. 93–105. Discusses using software metrics to allocate test efforts among a system's modules. A more extensive treatment of the data discussed in this article.

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Pink noise

By Frank D. Greco

Microsoft is trying its best to convince us subliminally that its new-generation operating system, which the marketing types are calling NT ("New Technology") or Windows NT, is a completely different animal from the beleaguered OS/2. Do you suppose Bill Gates invested all that time and money on development of OS/2 only to write it off? Let's face it—Windows NT is OS/2. Microsoft hopes that juxtaposing the words *Windows* ("success, popularity") with *NT* ("modern, techno-hip") will make us think that it has shaken off the ill effects of OS/2.

OLD WHINE, NEW BOTTLES

OS/2, the product, is dead, dead, dead (sorry, IBM—you're wasting your time). OS/2, the software, lives on, phoenix-like, in the form of Windows NT. It is not fathomable that Microsoft designed and implemented a completely new operating system, grafted Windows on top of it, made it portable to several different machine architectures, complied with the Posix standard (for humongous government bucks), and will be shipping it in mid-1992 without any influence from the failed OS/2. Windows NT is the wolf OS/2 in sheep's clothing.

Microsoft knows darn well the value of name recognition. Tandy never did and paid the price: it could never shake the stigma of the "Radio Shack" label. Did Tandy honestly believe that MIS managers would put their careers on the line and entrust the lifeblood of their companies to computers from "Radio Shack"? That's worse than buying terminals from "Teletype." Today's corporate purchaser or MIS manager most likely is a baby boomer, heavily influenced by television and very brand-conscious.

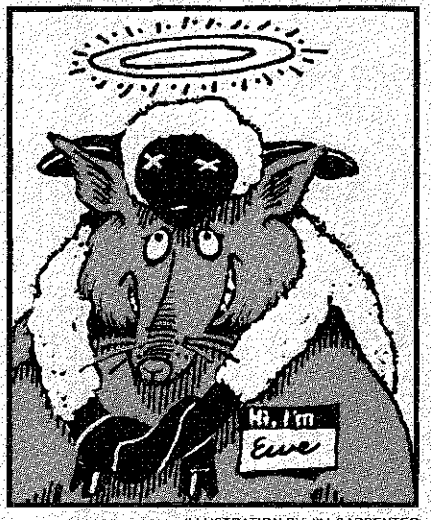


ILLUSTRATION BY JIM CARPENTER

Buying a piece of software called Windows NT shows the boss that you're a happenin' PC dude who's up there on the technological curve. Buying something called OS/2 shows that you're nowhere; it's as politically incorrect as eating red meat, smoking cigarettes, or failing to recycle.

ADOBE PLANS FOR POSTSCRIPT

Adobe has been trying to extend PostScript to handle multimedia documents. This future version of the fast-growing document standard is called Editable PostScript. Many popular applications produce PostScript to represent the resultant printed image in a procedural fashion. Unfortunately, most of these same applications cannot read PostScript, because there is no inherent structural sense to a PostScript document. Some time ago, Adobe released a document, called "structured PostScript," with commenting (as opposed to language) conventions, but this information is hardly enough to internally reconstruct the original document.

To rectify this "deficiency," Adobe chairman John Warnock has been announcing to everyone within earshot that Adobe will release a version of PostScript (i.e., Editable Postscript) that will be able to represent the internals of any document, complete with links to other documents. The result will allow one application to read a PostScript document from another application and maintain the document's structure. It would seem that a universal file format for hypermedia documents is on its way.

Judging from some research papers I unearthed, I would bet Adobe has two ace developers working on this product. Brian Reid and Paul Asente, both of whom implemented the precursor to the X Window System (called W), are now employed by Adobe. These gentlemen had extensive experience in designing structured documents at Stanford. One of Asente's papers, written while he was at DEC's Western Research Labs (and freely available), was called "Editing Graphical Objects Using Procedural Representations." Sounds like Editable PostScript to me. One caveat: Adobe isn't the only company working on such an animal.

MEAN GREEN(BACKS)

OK, we know Bill Gates has about 40 million shares of Microsoft stock and is worth about $4 billion. And we know that fellow founder Paul Allen has about 15 million shares and a paper value of $1.5 billion. With senior vice-president Steve Ballmer also owning 7 million shares ($750 million), that's a grand total of $6.25 billion among the three of them!

JUST THINKING

IBM: Think
Apple: Think Pink
IBM & Apple: Still Thinking
The rest of us: Think Again

GETTING BITS TO DEATH

Expect Fujitsu to start shipping its monster 2-gigabyte 5¼-inch hard disks by the last quarter of 1991. They're saying to expect prices around $3,500 for this gigantic disk drive. You should also expect these drives to be installed in a future Sun workstation—Fujitsu and Sun have a very close relationship (Fujitsu fabricated the first Sun SPARC chip).

Not to be outdone, Micropolis has announced the first 1-gigabyte 3½-inch hard drive with access times under 10 milliseconds. Expect to see these drives in the first or second quarter of 1992. We're not

Frank D. Greco is an independent consultant who offers industry analysis, systems consulting, and technological forecasting for Fortune 500 companies. Readers can contact Frank c/o Crossroads Technologies, Inc., P.O. Box 530, Fanwood, NJ 07023; (908) 754-7820. He can be reached via Internet at frank5@mercury.com.

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

too far from having a gigabyte of disk storage in a laptop (my Toshiba 5200 laptop already has 200MB). It won't be long before we see 100MB in a notebook!

A SPARC IN YOUR POQUET?

Why was the Poquet technical manager a bit ruffled when I asked about SPARC-powered Poquet handhelds at the last PC Expo in New York? Now that Fujitsu (the new Poquet owner) has a very large foot in the SPARC door, I expect to see some cross-pollination. We'll see either a very fast Poquet machine (imagine doing significant Lotus scenario-playing on a handheld!) or a SPARC notebook (using Poquet miniaturization techniques with SPARC power). In any event, Fujitsu will be shipping some very interesting SPARC machines stateside any day now.

dBASE OF dDEAL

I don't believe that the $440 million purchase of Ashton-Tate by Philippe Kahn and Borland was done just for dBASE. Kahn wants two things: Interbase (the Ashton-Tate subsidiary) and direct access to the dBASE customer—in that order. Borland is interested in Interbase because it offers an SQL client/server relational database with an object-oriented flavor. As the corporate world moves toward networks, client/server architectures, and object-oriented technology (the paradigm of the nineties), I'm sure Kahn views Interbase as the next Oracle (success in the minicomputer market).

There is much money to be made in this market. The only other database that was designed from its inception using a client/server architecture is Sybase. Sybase, however, is extremely expensive, leaving itself open to low-end Borland pricing. In addition, Sybase has been lethargic in converting its user interface tools to MS Windows and X Window. Kahn knows that momentum is building for Microsoft's SQL Server (developed by Sybase and practically identical to Sybase's workstation product). The hard part is that Borland must now consolidate the Paradox and dBASE product lines (my guess is that dBASE is a dying star) without alienating dBASE customers.

By the way, Ashton-Tate's selling price is about one-tenth of Bill Gates's paper worth.

VIDEO SUNRISE

Sun executives have been projecting that the Sun workstations that will be announced in 1992 will have built-in ISDN ports, Ethernet video telephone capabilities, and Intel DVI chips for multimedia. It appears that Intel's C i750 DVI chip might make it onto the Sun motherboard by 1993, and video will be a standard part of every Sun workstation. Although this does seem exciting, it is rather out of character for Sun to choose a proprietary chip to perform video processing. I sense that Sun has something up its chip-licensing sleeve.

MORE MODEM FOR THE MONEY

Have you noticed that 9600-baud modems are now hovering around the $300–$400 range? Make sure any modem purchase includes V.42 bis and MNP level 5.

If we called 'em *demodulators/modulators*, would modems then be called *demos?*

PERISCOPE

Professional Software and Hardware-Assisted Debuggers for 80386 & 80486 Systems

JUST RELEASED

Model IV for 386SXs

OS/2 Driver Support

32-Bit Toolkit

Rent Model IV

Windows 3 Driver Support Soon!

If you need a debugger but have no room left in the lower 640K of your 386™ or 486™ system, Periscope now has a new SOFTWARE-ONLY solution for you — Periscope/EM.

- Periscope/EM takes advantage of the extended memory you already have, rather than using any memory in the lower 640K.
- No run-away program can corrupt Periscope/EM because it's write-protected.
- You don't have to use a slot since there's no board to install.
- You can add an optional break-out switch to recover from crashes.
- Periscope/EM costs only $295, half the cost of Periscope Model I, with most of the same functionality.

Periscope/EM requires 386MAX® or BlueMAX®, version 5.11 or later, a 386 or 486 system with about 300K of extended memory, 32K of memory between 640K and one megabyte, DOS 5.0 or later.

New Debugging Tools

"I needed a means to debug interrupt handlers where I could really see what was going on. The hardware trace buffer is great. I was able to debug code in 3 days that I have been trying to debug for months!"

...writes Peg Sestrich with Prime Computer, on why she chose Periscope Model IV.

If you need more than your current software-based debugger gives you, Periscope Model IV, with these advanced NEW features, may be just what you're looking for.

- Supports 386 and 486 systems running up to 33MHz with its ICE-like capabilities.
- Real-time hardware trace buffer holds up to 16K CPU events.
- Enhanced analysis and display of trace buffer makes you more productive.
- Periscope/EM functionality, built-in at no extra cost, saves you money.

Just call toll-free 800/722-7006 for details or to order.

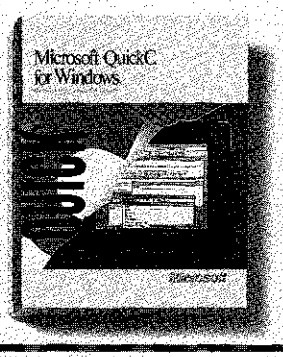
Real-time Periscope Model IV, shown with new 33MHz board and 486 pod.

The Periscope Company, Inc.

1197 PEACHTREE ST.
PLAZA LEVEL
ATLANTA, GA 30361
404-875-8080
FAX 404-872-1973

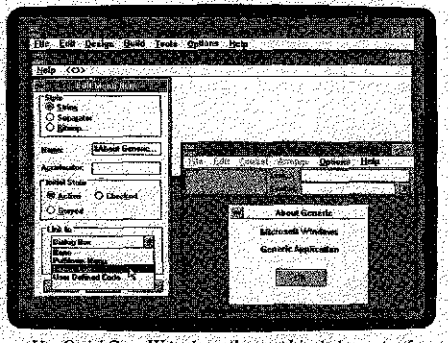
Circle 9 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

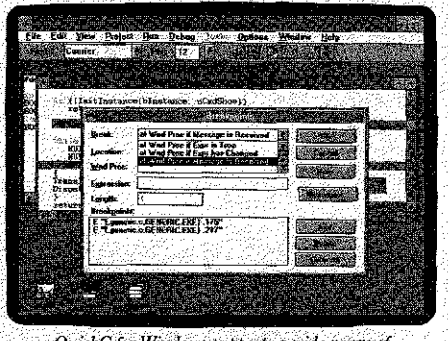


MICROSOFT SOFTWARE DEVELOPMENT TOOLS

Graphic proof that developing Windows apps is now easier.



Use QuickCase:W to draw the graphical elements of your Windows program, from menus and dialog boxes to cursors and icons. Then, let QuickCase:W generate commented expert-level C source code for the design that you've created.



QuickC for Windows supports a wide range of breakpoint types, from breaking at a location to breaking on a Windows procedure when a message is received.

Take a look at Microsoft® QuickC® for Windows™.

It's Windows-hosted, so you can edit, compile, and debug inside a single environment. Click on the Toolbar™ to choose frequently-used functions, from changing fonts to setting breakpoints. Workspace templates let you save your screen layouts, so you can reload them quickly from another session.

Plus, a remarkable tool known as QuickCase:W lets you create every element in your user interface with a few strokes of a mouse. Then it automatically generates the C source code you need.

All of which lets you create more applications. And more kinds of applications: C programs that call Windows APIs, graphical front-end programs for FORTRAN and COBOL, and C DLLs for other programs.

We suggest a visit to your Microsoft dealer. Because with QuickC for Windows, seeing is believing.

*© 1991 Microsoft Corporation. All rights reserved. Printed in the U.S.A. Inside the 50 United States, call (800) 541-1261, Dept. R25; outside the 50 United States, call (206) 936-8661. In Canada, call (416) 568-3503. Microsoft, CodeView, MS-DOS, QuickC and the Microsoft logo are registered trademarks and Windows, and Toolbar are trademarks of Microsoft Corporation. *As used herein, "DOS" refers to MS-DOS and PC-DOS operating systems.*

Key Features

- Windows-hosted integrated development environment including an editor, compiler, and debugger.
- All you need to write a Windows-based program.
- QuickCase:W generates source code from your program design and regenerates the code if you change the design.
- Wide range of breakpoint support, including breaking at a location, breaking when an expression has changed, and breaking at a Windows procedure when a message is received.
- Complete printed and online documentation on the Windows API.
- Toolbar for quick access to frequently used editing and debugging functions.
- Save Workspace and Load Workspace for saving and restoring window positions and breakpoints.
- Syntax highlighting in editor.
- Customizable tools menu allows you to run any Windows or DOS* program from within the QuickC for Windows environment.
- QuickWin library for converting well-behaved C programs for DOS to Windows programs.
- Generates Windows Executables, Windows Dynamic Link Libraries, QuickWin Executables for the Windows environment, and MS-DOS® Executables.

PROGRAMMER'S TIPS

- *To rebuild your character-based DOS applications to run under the Windows environment, select the "QuickWin EXE" Project type in the Options menu.*
- *Use the new and improved Dialog Editor to quickly and easily design dialog boxes for your programs.*

Microsoft

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Contact this advertiser directly

A tiny Windows class library

Writing Windows programs can be child's play with this simple C++ library

By John Dimm

Programmers don't like the feeling of losing ground. If you've worked in Windows long enough to be comfortable with the Software Development Kit's peculiar ways of doing things, you're probably regarding the switch to an object-oriented compiler and class library with caution, if not dread. The transition to C++ takes time, and there's no guarantee that you won't simply be trading in one set of quirks for another.

What you need is a simple library you can use right away. It turns out that a few pages of C++ code are enough to produce dramatic results.

SHRINKING GENERIC

The safest way to write a new Windows program is to start with a copy of GENERIC.C or some other SDK example. Windows programming is based on a lot of boilerplate code, and it would be crazy to type it all in each time. The SDK version of GENERIC, minus the About box, consists of close to 100 lines of source code. Using the following library, you can rewrite it in eight lines:

```
#include windows.h
#include "winclass.h"
#include "initinst.h"
BOOL InitInstance()
{
 static MAINWINDOW generic;
 return generic.Make();
}
```

This is not a code fragment but an entire Windows program. Compile and link it with WINCLASS.LIB to make a program that produces an empty window.

One goal of OOP is to minimize the duplication of general-purpose code in various source files. Reducing the size and complexity of GENERIC reduces the size and complexity of the programs you derive from it.

FOLLOW THAT MOUSE

For a more interesting example, Listing 1 is a program that causes the mouse to draw a line tracing its movements while the left button is held down.

Again, this is a complete Windows program. Let's look at how it works. SCRIBBLE responds to three different Windows messages: WM_LBUTTONDOWN, WM_MOUSEMOVE, and WM_LBUTTONUP.

LButtonDown is called whenever you press the left button of the mouse. SCRIBBLE gets a device context and sets the graphics current position to the mouse position. Windows packs the mouse position into a double word and passes it in the **lParam;** the inline functions **getx** and **gety** return the mouse position by unpacking it from **lParam.**

MouseMove is called whenever the mouse moves from one pixel location to another. If the handle to the device context is nonzero, SCRIBBLE draws a line between the last position and the new one. Of course, the mouse's actual path is probably not a straight line. The discrepancy between the real path and the screen trace becomes more obvious as you move more quickly. You can bring up a copy of Paintbrush next to the SCRIBBLE window to compare their performance.

When you release the mouse button, **LButtonUp** is called. SCRIBBLE releases the device context and sets the handle to the device context to zero, so that **MouseMove** is no longer activated to draw lines.

Note that no code is needed to call these functions. All three message handlers in SCRIBBLE are called automatically in response to the corresponding Windows messages (see Listing 2). That's the beauty of virtual functions and late binding: you write a function and give it a specific name, and it gets called at the right time. In standard C, whatever function you call **main** is executed at program startup. In C++, you can set up your own automatically executed functions.

In Windows parlance, WINCLASS turns the individual message handlers

John Dimm works on multimedia applications for Britannica Software in Solana Beach, California. He can be reached at 6075 Cozzens Street, San Diego, CA 92122, or on CompuServe at 70632,3444.

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

into automatically registered call-back functions.

LOOKING THROUGH WINDOWS

Now that we've seen some examples, let's look at what's going on behind the curtain.

The first thing to explain is how anything happens at all. Windows programs begin by executing **WinMain,** much as C programs execute **main.** GENERIC has no **WinMain.** In fact, it has only one function, **InitInstance.** Windows pays no attention to a function with that name.

WinMain still exists, but it's been moved into WINCLASS (see Listing 3). The idea is to transfer as much code as possible out of GENERIC and into WINCLASS. On the face of it, **WinMain** is an unlikely candidate. It has three main tasks:

- To perform application-specific initialization if there are no previous instances of the application, usually by registering classes.
- To perform instance-specific initialization, usually by making a main window.
- To enter a Windows message loop.

Although the last task is almost always coded the same way, the other two vary by application, so moving **WinMain** poses a problem. To keep code duplication at a minimum, register pointers to the application- and instance-specific initialization functions using WINCLASS. Windows calls **WinMain,** which in turn calls the initialization functions.

GENERIC's **InitInstance** is registered in the header file INITINST.H:

```
static BOOL InitInstance();
static void StartInstance()
{
 RegisterInstance( InitInstance );
}
#pragma startup StartInstance
```

#pragma startup causes **StartInstance** to be called before **main.** It would be convenient to execute **InitInstance** this way, but it's too soon: Windows hasn't yet passed an instance handle for the application through **WinMain.** Because **CreateWindow** needs the instance handle, **StartInstance** cannot make a window—but it can register a pointer to a function that will make one. By convention, this function is called **InitInstance.** Later, when Windows calls **WinMain,** the function pointer can be used to call GENERIC's **InitInstance** (see Listing 4).

WINCLASS has storage for 32 such functions. Usually, only one is needed, but this capacity allows you to make a program with new windows just by linking several object files. For example, you can make a program that produces both a SCRIBBLE window and a GENERIC window just by linking SCRIBBLE.OBJ, GENERIC.OBJ, and WINCLASS.LIB. The two **#pragma startup** statements register two static functions called **InitInstance,** one for each module. **WinMain** calls them in the order in which they were linked. Each module is unaware of its possible coexistence with others, and WINCLASS knows nothing about the details of any of the modules.

You can use the same scheme to define functions called **InitApplication. WinMain** always calls the **InitInstance** functions. When no previous instance of the application is found, **WinMain** calls any registered **InitApplication** functions.

Because it's the same for each program, the code for registering the functions is in a header file. It could just as well be copied into the source files, but putting the code into a header file makes programs look a bit simpler and enforces a useful naming convention.

THE KEY TO THE FRAMEWORK

After **WinMain** saves the application instance handle, it calls **InitInstance.** In the case of GENERIC, this function defines a static MAINWINDOW object called **Generic** and then calls the member function **Make** (see Listing 5).

Make contains calls to **RegisterClass** and **CreateWindow.** Combining these calls eliminates the need to distinguish the first time you make a window from subsequent times. It's also possible to register the class in **InitApplication,** following standard Windows practice. The **RegisterClass** call is preceded by a call to the virtual function

ILLUSTRATION BY BRENT BURKETT

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

DefineWClass, which sets up a default WNDCLASS structure. You can override **DefineWClass** and modify these defaults. Similarly, the call to **CreateWindow** is preceded by a call to the virtual function **DefineCreate.**

The parameters set by these virtual functions are the key to the whole class framework. The WNDCLASS structure identifies **ExportWndProc** as the Windows message handler for this class. Because every class is registered in the same way, they all use the same **ExportWndProc** (see Listing 6, which shows a technique similar to one found in Borland's WHELLO).

This registration scheme would be impossible in C, where each window class must have its own **WndProc.** How does **ExportWndProc** distinguish among different window classes? The answer is contained in the **CreateWindow** call.

A pointer to the object that made the call is passed through **CreateParams. ExportWndProc** gets the pointer when it receives the WM_CREATE message and places it in the extra bytes allocated for the window. As a result, each window has a pointer to the C++ object associated with it. You also save the window handle, **hWnd,** in the object itself. The window has a pointer to the object, and the object has a handle for the window. You can go back and forth at will.

MASSAGING THE MESSAGE

When a message comes in, Windows calls **ExportWndProc,** sending it the window handle and the parameters **iMessage, wParam,** and **lParam.** First **ExportWndProc** extracts the object pointer using the window handle. Then it calls one of the object's member functions, **WndProc,** which is defined in the base class WINDOW. It looks like a normal Windows message handler, with the typical big switch statement (see Listing 7).

Here's where it gets tricky. **WndProc** calls a different function in response to each message. But they're virtual functions—not ordinary functions—so the result depends on which class the object associated with the call belongs to. In the case of GENERIC, none of the virtual message-handling functions is overridden, so the base versions, which do nothing, get called. In the case of SCRIBBLE, three functions are overridden, so they get called instead.

And that's it. Windows calls **ExportWndProc,** which gets the object pointer from the window handle and then calls the object's **WndProc.** C++ knows which class the object belongs to and calls the corresponding virtual message

Deriving additional classes

Modal dialog boxes require some modification (see Listing 8). The **WndProc** for a dialog box is similar to the **WndProc** for a normal window; the differences can be handled through virtual functions.

The DIALOG class is derived from the WINDOW class, but instead of calling **DefaultWndProc** for unprocessed messages, you let Windows handle them. WINCLASS calls a virtual function, **DefaultProc,** rather than calling **DefaultWndProc** directly. The WINDOW class has a **DefaultProc** that calls **DefaultWndProc.** The DIALOG class replaces **DefaultProc** with a null function.

To demonstrate another way of binding the C++ object to the window, the listing uses properties, rather than window words, to store the pointer to the object. The result is the same. A more general solution would involve creating a new WINDOW class for the dialog so that window words could be allocated and used.

Because Dialog messages are passed through the WM_COMMAND message, each dialog box has to supply a **Command** function, where the real work goes on.

The same techniques can be used to encapsulate modeless dialog boxes, scrollbars, edit controls, and so on. In general, whenever you have the urge to copy a block of code from one source file to another, think about using an object with virtual functions instead. The base object embodies the structure of the code, and the virtual functions define the special features of each instance.

Another C++ technique is using constructors and destructors to simplify graphic device interface (GDI) functions. For example, to use a pen in a device context, you must

- Create a new pen;
- Select the new pen, saving the original pen;
- Use the new pen;
- Select the original pen back into the device context; and
- Delete the new pen.

You can do most of this with an object:

```
class HPENCLASS
{
HPEN hPen;
HPEN hOldPen;
HDC hDC;
public:
  HPENCLASS(HDC hDCPrm, int nPenStyle, int
          nWidth, COLORREF crColor)
    {
    hDC = hDCPrm;
    hPen = NULL;
    hOldPen = NULL;
    if (hDC)
      {
      hPen = CreatePen (nPenStyle, nWidth,
                        crColor);
      if (hPen)
        hOldPen = SelectObject (hDC, hPen);
      }
    };
  ~HPENCLASS()
    {
    if (hDC && hOldPen)
      SelectObject (hDC, hOldPen);
    if (hPen)
      DeleteObject (hPen);
    };
};
```

When you declare a PENCLASS object, it is selected automatically into the device context. When the object goes out of scope, it takes care of its own destruction and returns the device context to its former state. You don't have to pair the statements to create and select each pen with a few lines to deselect and destroy it. One declaration replaces four source lines.

You can do the same thing for brushes, palettes, device contexts, memory blocks, files, and so on. Every Windows object can be wrapped up in a C++ object. With a little work, you can arrange them in a useful hierarchy. It's good preparation for the more difficult task of using object-oriented design principles in your own code.

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
//  Drawing with the mouse
#include <windows.h>
#include "winclass.h"
#include "initinst.h"

class SCRIBBLE : public MAINWINDOW
{
  protected:
    HDC hDC;

    LPSTR GetClassName() {return "Scribble";};
    void MouseMove ();
    void LButtonDown ();
    void LButtonUp ();

  public:
    SCRIBBLE () { hDC = NULL; };
};

// When the user presses the left mouse button, get a device context and
// set the current position.
void SCRIBBLE::LButtonDown ()
{
  hDC = GetDC (hWnd);
  if (hDC)
  {
    MoveTo (hDC, getx(), gety());
    ShowCursor (FALSE);  // Turn off the mouse arrow.
    SetCapture (hWnd);   // Capture the mouse.
  }
}

// When the mouse moves from one pixel position to another,
// draw a line if the device context is set.
void SCRIBBLE::MouseMove ()
{
  if (hDC)
    LineTo (hDC, getx(), gety());
}

// When the left mouse button is released, release the device context.
void SCRIBBLE::LButtonUp ()
{
  if (hDC)
  {
    ReleaseDC (hWnd, hDC);
    hDC = NULL;
    ShowCursor(TRUE);   // Turn the mouse arrow back on.
    ReleaseCapture();   // Release the mouse.
  }
}

// Create a scribble window.
BOOL InitInstance()
{
  static SCRIBBLE scribble;
  return scribble.Make();
}
```

LISTING 2

```
// The WINDOW class produces a simple generic window:  overlapped, thick-
// framed, menuless, empty. Although there's no good reason to, it can be
// instantiated, as in GENERIC.CPP. The parameters wParam and lParam are
// copies of what Windows passes to the WndProc. Each member function
// parses them to retrieve whatever it needs. A more ambitious solution
// would be to parse them in the base class, passing digested and typed
// parameters to each message handler.
// class WINDOW
{
  protected:
    HWND hWnd;
    LPPAINTSTRUCT lpPaint;
    WORD wParam;
    LONG lParam;

    virtual void DefineCreate ( CREATESTRUCT &CreateStruct );
    virtual void DefineWClass ( WNDCLASS &WndClass );
    virtual LPSTR GetClassName () { return "Generic"; };
    virtual LPSTR GetWindowName () { return GetClassName (); };
    virtual LONG GetStyle () { return WS_OVERLAPPEDWINDOW; };
    virtual LONG GetParent () { return NULL; };
```


Innovative Developer's Tools!

MultiSpawn *New*

Instantly convert your application menu into a Multi-tasking menu! MultiSpawn allows your users to suspend the current application by pressing a hot key. The user can then start a new application from the menu, or return to a program already in progress. To use MultiSpawn, simply replace the *spawn*, *exec* or *system* function calls in the menu with the MultiSpawn call. **MultiSpawn also allows you to run large programs within another program by completely swapping out the original program.** *($149)*

TE Developer's Kit *Updated*

Incorporate text editing features into your application easily and cost effectively. Features: multiple files/windows, word-wrap and printing, edits files larger than available memory, reconfigurable keyboard commands and screen colors, undo, cut/paste, column block, and search/replace. **The kit also includes TER small editor routine. TER is a subset of TE. Your application calls this routine by specifying the window location and size, maximum file size, and an input buffer or file.** Supports Microsoft and Borland 'C' compilers. Includes the complete source code. **(Version: 3.0**, *DOS: $169, Windows: $269)*

Now also available version 3.5 that incorporates additional word processing features such as **bold**, underline, *italic*, line centering, and printer control. **(Version 3.5**, *DOS: $249, Windows: $349*)

WinMem: *Windows 3 Memory Manager*

WinMem provides an unlimited amount of virtual memory even in Windows' Real and Standard modes. It provides an unlimited number of global memory handles, and reduces the overhead per block to 4 bytes. Includes the complete source code. *($139)*

Spell Time

Spell Time consists of a dictionary (over 100K words) and the routines to access the dictionary. It also includes an application specific dictionary, and a user dictionary. The routine will suggest alternates for a misspelled word. Highly optimized: 50 to 75 words/sec on a 12 Mhz 286 computer. Uses virtual memory on disk, or EMS. An interface with TE included. Includes the complete source code. Supports Microsoft and Borland 'C' Compilers. *($369)*

Sub Systems, Inc.

1-800-447-6819

Fax: 1-617-438-0311

159 Main Street, #8C, Stoneham, MA 02180, (617)-438-8901

Circle 10 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    BOOL Show () { return ShowWindow ( hWnd, GetCmdShow () ); }
    void Update () { UpdateWindow ( hWnd ); }
    virtual void CallPaint ();
    virtual int GetCmdShow ();
    int getx () { return LOWORD (lParam); };
    int gety () { return HIWORD (lParam); };

    virtual void Create () {};
    virtual void InitDialog () {};
    virtual void Paint () {};
    virtual LONG Command () { return TRUE; };
    virtual void Timer () {};
    virtual void Size () {};
    virtual void MouseMove () {};
    virtual void LButtonDown () {};
    virtual void LButtonUp () {};
    virtual void Destroy ();
    virtual LONG DefaultProc (WORD iMessage)
        { return DefWindowProc ( hWnd, iMessage, wParam, lParam ); };
};

  public:
    WINDOW () { hWnd = NULL; };
    HWND GetHandle () { return hWnd; }
    void PutHandle ( HWND hWndPrm ) { hWnd = hWndPrm; }
    virtual BOOL Make ();
    ~WINDOW () { if (hWnd) DestroyWindow (hWnd); };
    LONG WndProc ( WORD iMessage, WORD wParamPrm, LONG lParamPrm );
};
typedef WINDOW * PWINDOW;

//  A main window brings down the application when it is closed.
class MAINWINDOW : public WINDOW {
  public:
    void Destroy () { WINDOW::Destroy ();
                      PostQuitMessage ( 0 ); };
};
typedef MAINWINDOW * PMAINWINDOW;

// A dialog box is created a different way. Its default proc does nothing.

class DIALOG : public WINDOW
{
  public:
    virtual LPSTR GetTemplateName () { return "GenericDlg"; };
    int Make (HWND hParent);
    LONG DefaultProc (WORD iMessage);
    void Destroy ();
};
typedef DIALOG * PDIALOG;
```

LISTING 3

```
//  WinMain: Main Windows function
#pragma argsused
int PASCAL WinMain ( HANDLE hInstance,
                     HANDLE hPrevInstance,
                     LPSTR lpszCmdLine,
                     int nCmdShow )
{
  MSG msg;

  Main.Init (hInstance, hPrevInstance, nCmdShow);

  if ( ! hPrevInstance )
    if ( ! Main.InitApplication () )
      return FALSE;
  if ( ! Main.InitInstance () )
    return FALSE;

  while ( GetMessage ( &msg, NULL, 0, 0 ) )  {
    TranslateMessage ( &msg );
    DispatchMessage ( &msg );
  }
  return msg.wParam;
}
```

LISTING 4

```
//  Main stores the parameters passed to WinMain and stores the
//  InitApplication and InitInstance procs for each module.
#define MAX_WINCLASSES 32
struct MAIN {
  BOOLPROC InitInst[MAX_WINCLASSES];
  BOOLPROC InitApp[MAX_WINCLASSES];
  int nApp;
  int nInst;
  HANDLE hInstance;
  HANDLE hPrevInstance;
  int nCmdShow;

  MAIN () { nApp = 0; nInst = 0; };
  BOOL InitApplication ();
  BOOL InitInstance ();
  void Init ( HANDLE hInstancePrm, HANDLE hPrevInstancePrm,
       int nCmdShowPrm);
};
MAIN Main;

//  Store the parameters passed to WinMain.
void MAIN::Init ( HANDLE hInstancePrm,
                  HANDLE hPrevInstancePrm, int nCmdShowPrm )
{
  hInstance = hInstancePrm;
  hPrevInstance = hPrevInstancePrm;
  nCmdShow = nCmdShowPrm;
}

// Each module registers its InitInstance function with Main. The Init-
// Instance functions will be called after the instance handle is stored.
void RegisterInstance ( BOOLPROC InitInstance )
{
  if (Main.nInst < MAX_WINCLASSES)
    Main.InitInst[Main.nInst++] = InitInstance;
}

//  Call the InitInstance for each module.
BOOL MAIN::InitInstance ()
{
  int i;
  for (i=0; i<nInst; i++)
    if ( ! InitInst[i] () )
      return FALSE;
  return TRUE;
}

//  Do the same thing with the InitApplication functions.
void RegisterApplication ( BOOLPROC InitApplication )
{
  if (Main.nApp < MAX_WINCLASSES)
    Main.InitApp[Main.nApp++] = InitApplication;
}

//  Call the InitApplication for each module.
BOOL MAIN::InitApplication ()
{
  int i;
  for (i=0; i<nApp; i++)
    if ( ! InitApp[i] () )
      return FALSE;
  return TRUE;
}
```

LISTING 5

```
//  Register a class and create a window.
BOOL WINDOW::Make ()
{
  WNDCLASS WndClass;
  CREATESTRUCT CreateStruct;

  // Call the virtual function DefineWClass, allowing descendents of WINDOW
  // to override the default values for the window class.
 DefineWClass ( WndClass );
  // Register the class, if it's not already registered.
  if ( ! GetClassInfo (Main.hInstance,
                       WndClass.lpszClassName,
                       (LPWNDCLASS)&WndClass) )
    if ( ! RegisterClass ( (LPWNDCLASS) &WndClass ) )
      return FALSE;

  // Allow descendents of WINDOW to change the default values for
  // the create structure, by overriding DefineCreate ().
  DefineCreate ( CreateStruct );
  CreateWindow ( CreateStruct.lpszClass,
                 CreateStruct.lpszName,
                 CreateStruct.style,
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

handler. You can add specific code in the derived class to handle the messages you care about. All windows use the same **ExportWndProc** and the same **WndProc.**

DABBLING IN C++

The **WndProc** in WINCLASS has cases for a few common Windows messages, but it leaves out more than it covers. Other messages can be handled either by adding them to WINCLASS (so that they're available to all windows) or by overriding **DefaultProc** in the declaration of a particular window class. The replacement for **DefaultProc** would contain a **switch** statement like the one in **WndProc,** with a **case** for the new message.

In C, each new class fields the messages it wants by adding cases to the main switch statement. In the class library described here, each derived object handles specific messages by redefining the necessary message handlers. The net result is less code to write for each window class in your system.

WINCLASS.LIB doesn't relieve you of the burden of learning the SDK, but it does take care of some details once and for all. It also lets you get your feet wet with C++ at your own pace. With these tips and techniques, you can make Windows programming easier by rolling your own class libraries. PJ

Memory Problems Solved!

MS-DOS 'C' AND ASSEMBLER PROGRAMMERS - BREAK THE 640K LIMIT!

XMEM A complete virtual memory system! — $149
- Manages expanded, extended, and can swap to disk.
- Eliminates memory fragmentation problems.
- Optionally emulates Windows & Macintosh memory managers.

EMPTY SHELL Run any program from your program! — $169
- Shell to any program, leaves less than 10K resident!
- Saves and restores screen - text or graphics!
- Swaps your program to expanded memory, extended memory or disk.

MEMORY TOOLKIT Easy access to all memory! — $99
- Read and write expanded or extended memory with simple file like calls.
- Create multi-megabyte arrays in expanded, extended, or on disk.
- Complete set of calls to EMS and XMS managers.

To Order Call:
1•800•755•7973
Visa / MasterCard / C.O.D.
International: (609) 587-7973
Fax: (609) 587-9412

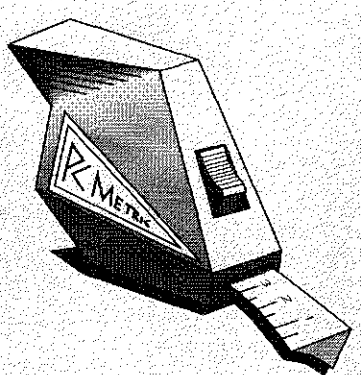
Kandu Inc.
"We Can Do What Others Can't"

ALL OUR PRODUCTS FEATURE: • Source code • Tested with Microsoft, Turbo, & Zortech C • 30 Day Money-back Guarantee • No Royalties! • 100% Assembler • Free technical support

Circle 11 on Reader Service Card

The Measure of a Great Program.

PC-METRIC™: The Measurement Tool For Serious Developers.



PC-METRIC is the software measurement tool that measures your code and identifies its most complex parts. So you can spend your time working in the areas most likely to cause problems.

PC-METRIC is fast, user-configurable, and includes a wide array of commonly accepted measurement standards.

Plus, versions of PC-METRIC are available to support virtually every popular programming language.

A Great Value By Any Measure.

PC-METRIC's price is only $199, and it comes with a 30-day money-back guarantee. Multiple user discounts are available, as well as site licenses and complete source code.

Order Now! Call (503) 829-7123.

COMPUTER LANGUAGE PRODUCTIVITY AWARD 1990

SET LABORATORIES, INC.
"Quality Tools For Software Craftsmen"
P.O. Box 868
Mulino, OR 97042
Phone: (503) 829-7123
FAX: (503) 829-7220

Contact this advertiser directly

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

If your Windows application needs images, you need ImageMan!

ImageMan provides a complete library for displaying and printing virtually any type of image. Our object-oriented approach allows your application to access all images in the same manner -- regardless of the type of image, your code doesn't change!

- Saves months of development effort.
- Use with any Windows 3.0 environment that supports DLLs, including: C, Pascal, Actor, Toolbook, Plus, etc...
- Completely supports TIFF, PCX, EPSF, WMF, and BMP images (with more to come....)
- Easily extended to support new image formats.
- Uses assembly language routines for *Speed.*
- Highly configurable - you have *full* control.
- Royalty Free!

Call today for free demo!

DTI Data Techniques, Inc.
1000 Business Center Drive, #120
Savannah, GA 31405

$395
$995 w/Source

800-868-8003
912-651-8003

Circle 12 on Reader Service Card

TIFF, PCX, & GIF SDK FOR WINDOWS 3.0

Add TIFF, PCX, or GIF support to your Windows 3.0 Application without learning to work with file formats!

TIFF SDK For Windows is a complete set of easy-to-use tools for adding TIFF 5.0 support to your application without learning the complexity of Tag Image File Format. Features include support for: Group 3 and Group 4 fax file formats, LZW and Packbit compression, 24-bit color images, chained images and device independent bitmaps. Includes full documentation plus a sample application complete with source code. **Only $299.95**

PCX SDK For Windows allows addition of the PCX file format to an application. Comes with full documentation, including a sample program with source code. **Only $149.95**

GIF SDK For Windows adds the latest GIF 24-bit color support to your application. Includes documentation plus a sample program complete with source code. **Only $149.95**

IMAGE SDK For Windows contains functions to read, write, and manipulate Microsoft Windows CLP, WMF, BMP, and MSP file formats. Zooms, inverts, & rotates bitmaps. Comes with documentation, sample program and source code. **Only $199.95**

CALL for info on NEW GrabMe Capture Utility
TEL: (914) 277-7006 FAX: (914) 276-8418
VISA/M.C./C.O.D/CHECK
BLACK ICE SOFTWARE INC.
Crane Road, Somers N.Y. 10589

Circle 13 on Reader Service Card

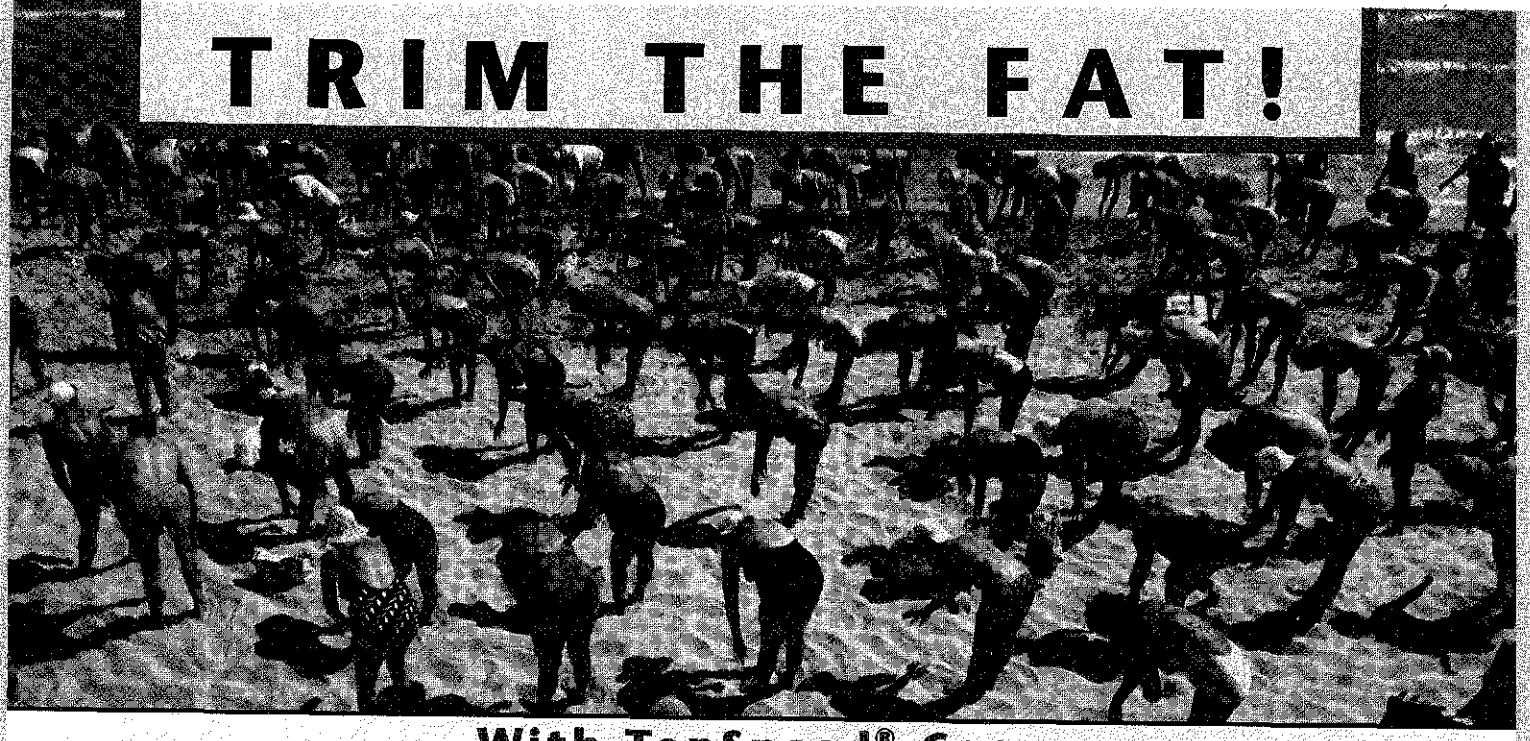

```
                CreateStruct.x,
                CreateStruct.y,
                CreateStruct.cx,
                CreateStruct.cy,
                CreateStruct.hwndParent,
                CreateStruct.hMenu,
                CreateStruct.hInstance,
                CreateStruct.lpCreateParams );

// The object's hWnd was set in ExportWndProc when WM_CREATE was processed.
  if ( ! hWnd )
    return FALSE;

  Show ();
  Update ();

  return TRUE;
}

//  default window class
void WINDOW::DefineWClass ( WNDCLASS &WndClass )
{
  WndClass.style         = NULL;
  WndClass.lpfnWndProc   = ExportWndProc;
  WndClass.cbClsExtra    = 0;
  WndClass.cbWndExtra    = sizeof ( PWINDOW );
  WndClass.hInstance     = Main.hInstance;
  WndClass.hIcon         = NULL;
  WndClass.hCursor       = LoadCursor ( NULL, IDC_ARROW );
  WndClass.hbrBackground = GetStockObject ( WHITE_BRUSH );
  WndClass.lpszMenuName  = NULL;
  WndClass.lpszClassName = GetClassName ();
}

//  default create structure
void WINDOW::DefineCreate ( CREATESTRUCT &CreateStruct )
{
  CreateStruct.lpszClass      = GetClassName ();
  CreateStruct.lpszName       = GetWindowName ();
  CreateStruct.style          = GetStyle ();
  CreateStruct.x              = CW_USEDEFAULT;
  CreateStruct.y              = 0;
  CreateStruct.cx             = CW_USEDEFAULT;
  CreateStruct.cy             = 0;
  CreateStruct.hwndParent     = GetParent ();
  CreateStruct.hMenu          = NULL;
  CreateStruct.hInstance      = Main.hInstance;
  CreateStruct.lpCreateParams = (LPSTR) this;
}
```

LISTING 8

```
//  Store pointer to window object.
inline void SetWndPointer ( HWND hWnd, PWINDOW pWindow )
{
  #if NEAR_POINTERS
    SetWindowWord ( hWnd, WW_POINTER, (WORD) pWindow );
  #else
    SetWindowLong ( hWnd, WW_POINTER, (LONG) pWindow );
  #endif
}

//  Retrieve pointer to window object.
inline PWINDOW GetWndPointer ( HWND hWnd )
{
  #if NEAR_POINTERS
    return (PWINDOW) GetWindowWord ( hWnd, WW_POINTER );
  #else
    return (PWINDOW) GetWindowLong ( hWnd, WW_POINTER );
  #endif
}

// See the warning in Borland's example WHELLO.CPP. We get the C++ window
// object pointer from the window extra bytes, which are zero-initialized
// by Windows. So we can assume any non-zero value is a pointer to a win-
// dow object. Unfortunately, the fact that extra bytes are zero-initial-
// ized is undocumented. If this changes in a future version of Windows,
// or if you don't like using undocumented features, you can store the
// pointer in a property rather than extra bytes. The property method is
// standard and is implemented below for dialog boxes.
inline BOOL IsWndPointer ( PWINDOW pWindow)
{
  return (pWindow != NULL);
}
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

TRIM THE FAT!



With TopSpeed® C++

The Only C++ Compiler with Smart Method Linking

C++ programs are notoriously large to begin with. Add in a third party library and you have obese executables. TopSpeed C++ with Smart Method Linking makes the overweight .EXE a thing of the past. Smart Method Linking removes unused data or methods (be they static or virtual) at the link stage, thus giving you the slimmest possible .EXE!

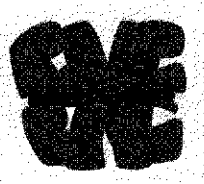
TopSpeed C++ is:

- The only C++ compiler with Smart Method Linking
- The only C++ compiler that implements AT&T 2.1
- The only C++ compiler with a fully configurable calling convention including register parameters
- The only C++ compiler compatible with Pascal and Modula-2 object formats
- The only C++ compiler supplied with a validated ANSI C compliant library
- The only C++ compiler that provides multi-tasking under DOS
- The only C++ compiler that supports DOS DLLs
- The only C++ compiler that checks compiler options as part of the make process

"We have graphics intensive, menu and event driven programs that require small executables. TopSpeed C++ gave us the best results. The DOS DLLs are fantastic!"
Dr. Arun Bommannavar, Research Scientist

"TopSpeed C++ is the best choice if you want fast and small programs. TopSpeed C++ clearly shows that C++ doesn't have to mean larger and slower programs."
Lawrence Widing, Senior Software Engineer

Before *with Borland C++*



Dr. Bommannavar's .EXEs	Before	After
Program #1 (DOS)	164,944 bytes	88,809 bytes
Program #2 (DOS)	168,480 bytes	111,877 bytes
Program #3 (DOS)	170,032 bytes	94,095 bytes
Lawrence Widing's .EXEs		
Program #1 (Windows)	166,400 bytes	74,300 bytes
Program #2 (DOS)	112,020 bytes	42,410 bytes

After *with TopSpeed C++*

TopSpeed® C++

THE END OF IDLE METHODS... THE END OF THE OVERWEIGHT .EXE

Standard Edition: $198 Professional Edition: $297 Master Edition: $594

Also available: TopSpeed Pascal with OOP, TopSpeed Modula-2 with OOP, TopSpeed C

All TopSpeed compilers support DOS, Windows and OS/2

To order call: (800)448-4440
TopSpeed Info Hotline: **(603)899-6470**
TopSpeed Fax Line: **(603)899-6480**
TopSpeed BBS: **(603)899-5751**

TopSpeed Consortium

JPI TopSpeed Consortium, Inc.
117 Hunt Hill Rd.
Rindge, NH 03461

Circle 14 on Reader Service Card

TopSpeed is a registered trademark of Jensen & Partners International, Inc.; Borland C++ is a registered trademark of Borland International, Inc.

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

"Without a doubt, OPUS Make is the hottest make utility on the market"
Tom Swan, PCWorld

OPUS SOFTWARE

OPUS Make

For professional programmers, OPUS Make is the superior choice in a make utility. It is the fastest most full-featured make utility there is. Features include:

- DOS version uses only 3K memory!
- Multiple directory support.
- Supports: Polytron PVCS™, Burton Systems TLIB™, Microsoft LIB™, SLR Systems OPTLIB™
- Generates automatic response files of unlimited length for LINK and LIB.
- OPUS MKMF included: An automatic dependency generator that fully understands C preprocessor directives.

Both DOS and OS/2 executables for only: $129 MC, Visa, COD, or P.O. 60 day MBG!

Call 1-800-248-OPUS
International: 415-864-7901
1032 Irving Street, Suite 439, San Francisco, CA 94122
Trademarks owned by their respective companies

Circle 15 on Reader Service Card

The Fast, Flexible Easy to Use GUI

TEGL WINDOWS TOOLKIT II

Programmers, now you can **create true event-driven stand-alone GUI** (graphical user interface) applications for DOS. Applications will not need any windows operating environment. Outstanding features include: local menus; resizeable windows; sliders; scrolling; mouse/keyboard/timer interrupt handlers; virtual memory manager that uses EMS/hard disk and handles; dialogue management; 49 fonts; world coordinates; Turbo Pascal objects and Turbo C++ class library.

Font and icon editors provide the tools to customize the finest details of your programs. TEGL Graphic Interface provides BGI compatibility with greater speed. **No runtime modules nor royalty payments** are required on applications developed with TEGL. The Toolkit adds as little as 100K to the size of an application. The **complete source code** is provided.

Supported graphics modes (colors): CGA(2), Hercules(2), EGA(16), VGA(16) and Super VGA(16, 256).

Supported compilers: Turbo Pascal, Stony Brook Pascal+, Turbo C®, Turbo C++®, Borland C®, Quick C®, Microsoft®C, WATCOM® C, and JPI Topspeed C.

****

TEGL WINDOWS TOOLKIT II IS ONLY $99. *ORDER NOW!*
Phone: (604) 669-2577 **Fax: (604)688-9530**

****

Shipping & Handling $10
($15 outside North America)
Visa, Mastercard, Cheque

TEGL Systems Corporation
780-789 West Pender Street
Vancouver, BC, Canada V6C 1H2

30 day money back guarantee

Trademarks are property of their respective holders.

Circle 16 on Reader Service Card

```
//  the WndProc called by Windows
LONG FAR PASCAL _export ExportWndProc ( HWND hWnd,
                WORD iMessage, WORD wParam, LONG lParam )
{
  PWINDOW pWindow = GetWndPointer ( hWnd );

  // If the window has already been initialized, call its WndProc.
 if ( IsWndPointer (pWindow) )
    return pWindow->WndProc ( iMessage, wParam, lParam );
  if ( iMessage == WM_CREATE ) { // The window is not yet initialized.
      // Extract the pointer to the WINDOW object, placed by
      //   DefineCreate.
    pWindow = (PWINDOW) ( (LPCREATESTRUCT)lParam)->lpCreateParams;

    // Store a pointer to the WINDOW object in the Window words.
 SetWndPointer ( hWnd, pWindow );

    // Store the window handle in the WINDOW object, so that it's
    // available when the object processes the WM_CREATE message.
 pWindow->PutHandle ( hWnd );

    // Pass the WM_CREATE message to the WINDOW object.
    return pWindow->WndProc ( iMessage, wParam, lParam );
  }

  // The window is not yet initialized.
  return DefWindowProc ( hWnd, iMessage, wParam, lParam );
}
```

LISTING 7

```
//  Handle Windows messages, calling the appropriate virtual
//    functions. Add cases as needed.
LONG WINDOW::WndProc ( WORD iMessage, WORD wParamPrm, LONG lParamPrm )
{
  LONG retcode = TRUE;

  // Save wParam and lParam so we don't have to pass them around as
  // parameters.
  wParam = wParamPrm;
  lParam = lParamPrm;

  switch (iMessage)  {
    case WM_CREATE:
      Create ();
      break;
    case WM_INITDIALOG:
      InitDialog ();
      break;
    case WM_COMMAND:
      retcode = Command ();
      break;
    case WM_PAINT:
      CallPaint ();
      break;
    case WM_SIZE:
      Size ();
      break;
    case WM_MOUSEMOVE:
      MouseMove ();
      break;
    case WM_LBUTTONDOWN:
      LButtonDown ();
      break;
    case WM_LBUTTONUP:
      LButtonUp ();
      break;
    case WM_TIMER:
      Timer ();
      break;
    case WM_DESTROY:
      Destroy ();
      break;
    default:
      return DefaultProc ( iMessage );
  }
  return retcode;
}
```

LISTING 8

```
//  Make a modal dialog box and wait until it's done.
int DIALOG::Make (HWND hParent)
{
  FARPROC lpfnDlg;
  int nResult;

  lpfnDlg = MakeProcInstance ( (FARPROC)ExortDlgProc, Main.hInstance);
  nResult = DialogBoxParam (Main.hInstance, GetTemplateName (),
                            hParent, lpfnDlg, (DWORD) this);
  FreeProcInstance (lpfnDlg);
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
  return nResult;
}

//  Store pointer to dialog object.
inline void SetDlgPointer ( HWND hDlg, PDIALOG pDialog )
{
  #if NEAR_POINTERS
    SetProp ( hDlg, WP_POINTER_OFF, (HANDLE)pDialog );
  #else
    SetProp ( hDlg, WP_POINTER_SEG, (HANDLE) FP_SEG (pDialog) );
  SetProp ( hDlg, WP_POINTER_OFF, (HANDLE) FP_OFF (pDialog) );
#endif
}

//  Retrieve pointer to dialog object.
inline PDIALOG GetDlgPointer ( HWND hDlg )
{
  #if NEAR_POINTERS
    return (PDIALOG) GetProp ( hDlg, WP_POINTER_OFF );
  #else
    return
      (PDIALOG) MK_FP ( GetProp ( hDlg, WP_POINTER_SEG ),
                        GetProp ( hDlg, WP_POINTER_OFF ) );
#endif
}

//  Remove the property used to store the dialog object pointer.
inline void DeleteDlgPointer ( HWND hDlg )
{
  #if NEAR_POINTERS
    RemoveProp ( hDlg, WP_POINTER_OFF );
  #else
    RemoveProp ( hDlg, WP_POINTER_SEG );
    RemoveProp ( hDlg, WP_POINTER_OFF );
  #endif
}

//  the DlgProc called by Windows
LONG FAR PASCAL _export ExportDlgProc ( HWND hDlg,
                WORD iMessage, WORD wParam, LONG lParam )
{
  PDIALOG pDialog = GetDlgPointer ( hDlg );

  // GetDlgPointer will return zero if the dialog is not yet initialized.
  if ( pDialog )
    return pDialog->WndProc ( iMessage, wParam, lParam );

  if ( iMessage == WM_INITDIALOG ) {
    pDialog = (PDIALOG) lParam;

    SetDlgPointer ( hDlg, pDialog );
    pDialog->PutHandle ( hDlg );

    return pDialog->WndProc ( iMessage, wParam, lParam );
  }
}

// Remove the property used to store the pointer to the C++ dialog
// object.  From here on, GetDlgPointer will return zero.
void DIALOG::Destroy ()
{
  DeleteDlgPointer (hWnd);
  hWnd = NULL;
}

//  For unprocessed messages to dialog boxes, return FALSE.
#pragma argsused
LONG DIALOG::DefaultProc (WORD iMessage) { return FALSE; }
```

Editor Headaches?
Trade in Your BRIEF!
SWITCH to the COMFORT and FREEDOM of
Multi-Edit
At a SHOCKING
50% OFF!
Admit it. You've been just a little bit curious about Multi-Edit, right? The programmers' editor that everyone is talking about! Unparalleled configurability. Incredible ease of use. A list of standard features that blows the socks off the competition! Well, here's your chance to find out for yourself that MULTI-EDIT CAN DO MORE FOR YOU THAN ANY EDITOR YOU'RE USING NOW! WE GUARANTEE IT!
"Mail us proof-of-ownership" of BRIEF, SAGE PROFESSIONAL, VEDIT PLUS, EPSILON, SLICK or CHEETAH and we will give you 50% off on the purchase of a Standard Multi-Edit. Apply the same dollar amount toward purchase of Professional.
(Mail the original proof-of-ownership sticker on the first page of the user manual for any one of the products named above. No photocopies, please!)
DO IT NOW!
Don't put it off! This offer is only good through 12/31/91.
TEL: (602) 968 1945
FAX: (602) 966 1654
1830 W. University Dr. • Suite 112
Tempe, AZ 85281
Still not sure...?
Get your FREE, FULLY FUNCTIONAL DEMO!
LIST PRICES $129. Standard $199. Professional
C.O.D.
AMERICAN CYBERNETICS
Multi-Edit is a trademark of American Cybernetics, Inc. Other trademarks are the property of their respective holders.
Circle 17 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Trouble in Object City

It's not too late to rescue OOP from the clutches of inflexibility

By Brett Glass

Object-oriented programming (OOP) seems to be taking the world by storm. Everyone is getting into the act—from ParcPlace and Digitalk, who have sold Smalltalk implementations for years, to Borland and Zortech, who charged into the fray with hot implementations of C++ well ahead of Microsoft. Apple, Microsoft, JPI—and, of course, Borland—offer object-oriented Pascals. We hear announcements of object-oriented versions of Fortran, Prolog, and COBOL (yes, COBOL!).

Programmers and organizations of all stripes are buying these products, thereby making substantial commitments to OOP. Clearly, it isn't just the "programming fad *du jour*"—OOP is here to stay.

OOP has many progenitors. Simula and Smalltalk were early pioneers; Object Pascal, various dialects of Lisp with "flavors" (i.e., classes), and Bjarne Stroustrup's C++ have since made their mark. Designers building new languages and environments have borrowed a concept here and added a twist there in search of the best object-oriented paradigm.

Whenever a technology emerges without a single, insightful leader—and sometimes even when there is one—trouble often follows. A case in point is the de facto standard architecture of the IBM PC and its descendants: four memory-management schemes (each promoted by a different vendor), eight video standards, three buses, and a limited number of non-shareable interrupt levels that invite conflicts, all make life interesting indeed for hardware and software designers, not to mention users.

After the fact, anyone can see the flaws in an architecture and propose good fixes, but hindsight comes too late once the standard has been established. That's why it's so important to take a careful, critical look at OOP now, in its relative infancy. This essay is intended to provide food for thought and discussion about the direction and implications of the object-oriented paradigm while there's still time to do something about it.

Brett Glass is a freelance programmer, author, hardware designer, and perpetual student residing in Palo Alto, California. Brett was one of the original architects of the IEEE 802.5 Token Ring LAN, and he co-authored Living Videotext's ThinkTank 2.0. Brett earned his MSEE from Stanford University and his BSEE from the Case Institute of Technology. He can be reached as [72267,3673] on CompuServe or as "rogue@well.sf.ca.us" on USENET and the Internet.

HANDCUFFED BY HIERARCHIES

One classic architectural mistake that crops up in both hardware and software design is limiting the arrangement of the components to an inflexible linear or hierarchical structure. *PC Magazine* columnist Jim Seymour's experiences when organizing his ideas with an outline processor provide a good example: "I use outlining programs to organize masses of raw ideas," he says. "I throw the ideas in as fast as I can type. I tuck some under other ideas to which they clearly seem subsidiary, and others I tuck in under *those*, to create hierarchies of ideas."

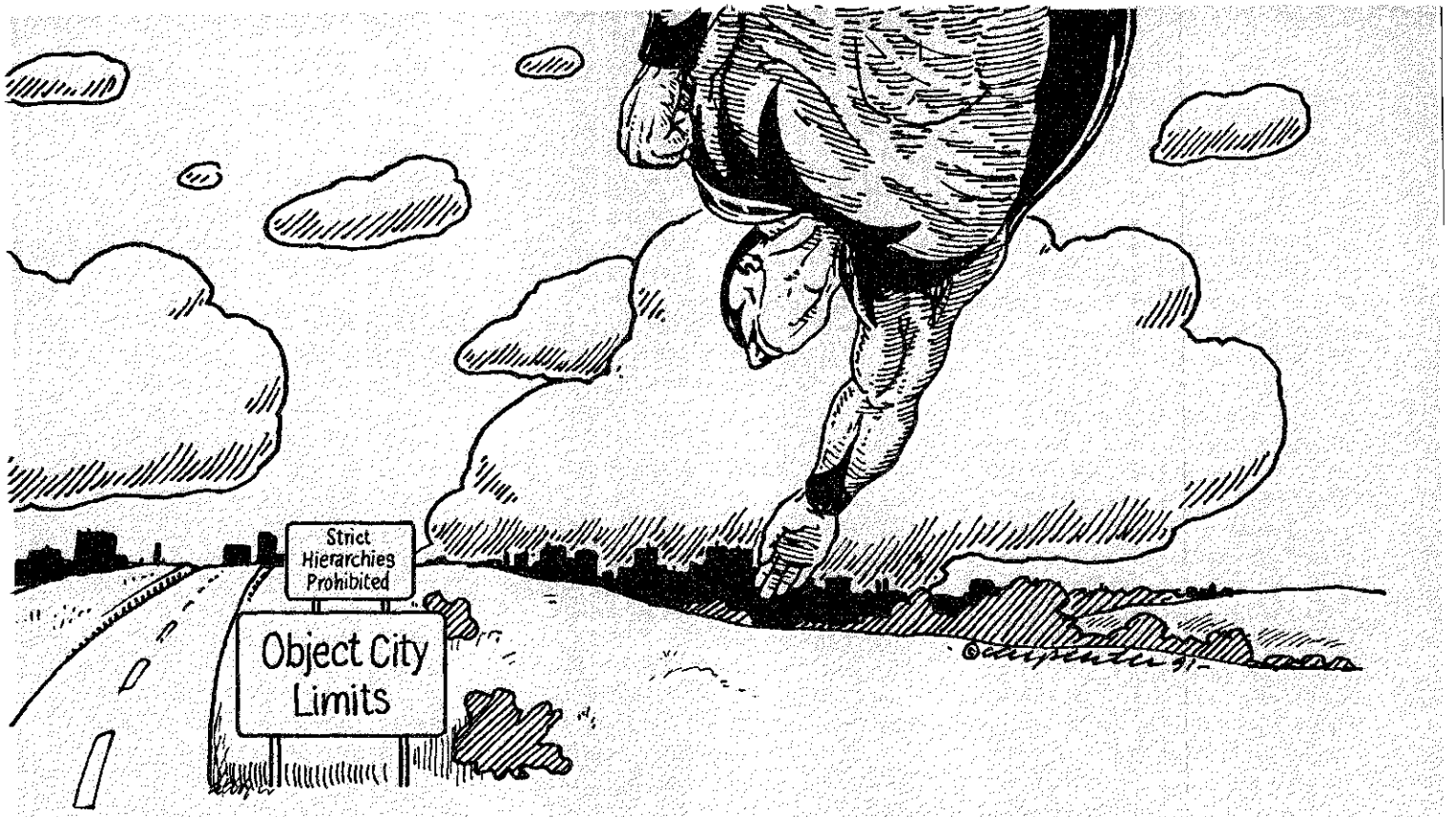
"But the minute I finish that list and read over it," says Seymour, "I realize just how wrong I was. There are complicated relationships—clear links and hierarchies—I never understood until I saw the finished list. So I start hauling things around, demoting this a level or two, pulling that from a third-level subtopic under Item G to a top-level heading all its own."

Seymour observes that the outline processor isn't designed to understand the many-way links between ideas. It therefore makes some kinds of rearrangement difficult and forces a hierarchical order onto groups of concepts that perhaps would be better arranged as a random network.

WHAT CAN'T BEND, BREAKS

It's tough to rearrange an outline, but it's even tougher to rearrange a hierarchy of objects in an object-oriented program, especially if a lot of sweat and code have been invested in the old structure. No matter how much foresight you apply or how carefully you plan, you will probably reach a point where you must rearrange

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



your hierarchies—or where no purely hierarchical structure will do.

Borland learned this lesson when it added the concept of *units* to its Turbo Pascal language. In Turbo Pascal 4.0, the units of a program had to be arranged in a strict hierarchy: if Unit B used data structures or procedures from Unit A, Unit A couldn't use anything from Unit B.

One of my programs had an Errors unit, which handled runtime errors, and a Files unit, which managed files. The Files unit needed to call the Errors unit to signal I/O errors; if the program terminated prematurely, the Errors unit needed to call the Files unit to close files. I managed to craft an assembly language "back door" to circumvent this roadblock, but there was no easy way to implement this seemingly natural relationship between units until TP 5.0—which, like Modula-2, allowed a "network" of units as long as certain pathological circularities did not exist.

What does all this have to do with OOP? Virtually all object-oriented systems rely on relatively strict hierarchies. (Multiple inheritance, as implemented in C++, bends the rules, but only a bit.) Inheritance is useful (see "The importance of good defaults," *PJ* 8.2), but more useful still would be a paradigm that allowed programmers to assemble new classes from a network of smaller components—in a sense, "modular" objects.

With more structural options at hand, we'd be less likely to suffer the pain and inconvenience of ripping up an existing hierarchy by the roots—or turning it inside out—to implement a new concept or feature. We'd also be less subject to another troublesome OOP problem, one that I have dubbed "the uninvited relatives syndrome."

OUT OF HOUSE AND HOME

Too often, when you want to use a class that's part of a relatively deep hierarchy, you'll find the program awash in code pertaining to its superclasses. Think of it this way: imagine you've invited a **friend** over for dinner. At the appointed time, he arrives—along with all his forebears: parents, grandparents, even the spirits of a few ancestors who are now, as it were, virtual. These ancestors stampede across the threshold and proceed to eat you out of house and home. Your **friend** explains, "Well, Ma taught me how to hold a spoon, so I need her here to help me eat. And Gramps taught me how to read, so he needs to be here in case I want to read anything. And Great-grandma . . ."

The thought of anyone dragging his ancestors with him everywhere he goes is comical, but having to put up with relatives of a class you import from a large toolkit isn't funny. Even with "smart linking," which cuts down on uninvited "sibling" classes, the overhead can be daunting.

Borland's Object Windows Library, a superb class hierarchy for Windows programming, shows the first symptom of "object glut": the hierarchy for even a simple program contains so many classes that their Virtual Method Tables begin to crowd other structures out of memory. Only by introducing Dynamic Method Tables—which sacrifice a small amount of speed for space—was Borland able to keep bookkeeping overhead at a reasonable level.

One of the touted advantages of OOP is the ability to use "toolkits" of objects from which you can select classes and create subclasses according to your needs. But have you ever tried to combine pieces of two toolkits, with two different hierarchies arranged two different ways? Here, the ability to assemble objects from smaller components in customized, structured networks would free programmers from the tyranny of monolithic hierarchies that must be imported wholesale, bag and baggage.

CLASH OF THE PARADIGMS

The Object Management Group, a large and growing industry consortium, is working on standards that, among other things, establish the behavior of objects that are created and manipulated via networks. Software that runs in such an environment must be flexible at runtime as well as at compile time; a program is likely to "discover" new classes of objects as it runs and will be expected to manipulate them appropriately.

The same is true in environments such as Windows, whose object linking and embedding (OLE) technology may reveal the

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

existence of a new class to your program when an object is "dropped" into its window. Penpoint, Go Corporation's sophisticated pen-based GUI, also allows classes to be created and destroyed at runtime.

Unfortunately, few of the current crop of object-oriented languages are dynamic enough to comprehend such environments. Xerox's Smalltalk can rebuild the innermost parts of the operating system as it is running (most current implementations share this flexibility to some degree), but the most popular object-oriented language, C++, is statically compiled and does not allow classes to be created or destroyed at runtime.

A C++ program that deals with a dynamic object-oriented environment must support two kinds of classes that have two kinds of semantics: C++ classes, which are manipulated via the fundamental constructs of the language, and the environment's classes, manipulated by the C-like, non–object-oriented constructs of the language. It's a shame when the powerful, object-oriented features of a language can't extend to the environment in which programs run. And it's a poor idea, as William of Occam once noted, to multiply entities beyond necessity—to deal with two kinds of objects that abide by different sets of rules.

The only way to accommodate dynamic, object-oriented environments without this duplication of effort is to reevaluate our language paradigms. Is it possible to use fully compiled code where practical and, at the same time, deal with a changing world in which new objects and classes come and go?

Before "strictly static" OOP languages proliferate, like the IBM PC architecture, to the extent that there's no turning back, we need to send our language designers back to the drawing board, exhorting them to seek new language models that gracefully accommodate both worlds. Otherwise, we will be left with tools that are largely extensions of static, non–object-oriented languages and that don't address our real needs in increasingly dynamic environments.

OOP, OOP, AND AWAY

None of these criticisms should be construed to mean that I don't like, or don't have high hopes for, object-oriented programming. OOP is far and away the most productive programming methodology I've encountered; my concerns stem from my desire to see it evolve into a solid base that supports, rather than limits, future generations of programmers.

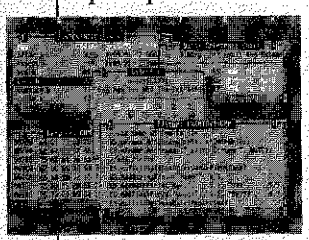
OOP is at that rare moment in the history of a new idea: gaining recognition, acceptance, and clout as it begins to produce tangible results. At this juncture, it's crucial that we steer future developments in the right direction. Only if we recognize the shortcomings of current OOP implementations and language paradigms and act on our experience can we develop object-oriented programming to its full potential—and stamp out trouble in Object City. PJ

Award Winning User Interface Classes Now for C++

Don't start from scratch with your text mode user interfaces! Save time with **Object Professional™ for C++**. Object Professional's 100 classes provide a windowing system with multiple overlapping and resizeable windows that include mouse support and scroll bars. Higher level objects provide ready-to-use text editors, dialog boxes, scrolling data entry screens, help systems, and more. The objects are highly flexible. You can create text-mode Windows look-alikes (SAA/CUA) or your own unique look and feel. All without event-driven programming.

Interactive Screen Builders

Object Professional includes utilities to help you build menu systems and data entry screens fast. Design and test interactively, then automatically generate source code.



Hot Demo Programs

Plenty of example programs get you up to speed fast. One is a desktop manager that includes text and file browsers, pick lists, modem dialer, and more. It provides you with classes you can reuse and illustrates how to build an ambitious application using Object Professional.

Source Code and No Royalties

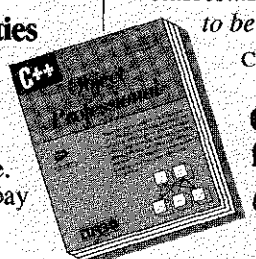
Object Professional includes complete documentation, pop-up help, free technical support direct from the authors by telephone and on CompuServe. You get full source code and pay no royalties.

An Award Winner!

Object Professional for Turbo Pascal was chosen as one of nine development products that "Jolted" the industry during 1990. Object Professional for C++ is the UI objects from that library ported to C++.

"Object Professional...a thoroughly object-oriented, beautifully documented, immaculately designed, comprehensive set of extensible classes...You really can't afford to be without this package."

Computer Language, April 1991



Object Professional for C++, only $249.

Call toll-free to order. 1-800-333-4160

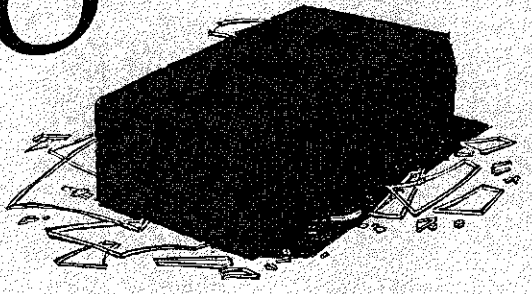
9AM - 5PM MST Monday through Friday, U.S. and Canada.
For more information call (719) 260-6641, fax to (719) 260-7151, or send mail to CompuServe ID 76004,2611
TurboPower Software PO Box 49009 Colorado Springs, CO 80949-9009
© TurboPower Software, 1991

Object Professional (OPro) for C++ requires Borland C++. Satisfaction guaranteed or your money back within 30 days. Add $5 per order for std. shipping in U.S. Inquire about other shipping options. Incl. dual media. Registered owners of OPro for TP may purchase OPro for C++ at a special price of $150.

Circle 18 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

BREAK into Windows Programming



We're breaking Windows programming wide open with **Windows Tech Journal,** the only magazine devoted exclusively to tools and techniques for Windows developers.

Windows programming demands new skills, new knowledge, and new tools. That's why we've packed **Windows Tech Journal** with authoritative reviews, enlightening articles, and insightful commentary by top Windows experts. **All the Windows information you need in one place.**

Inside every issue, you'll find feature articles on innovative Windows techniques, with complete source code. Our detailed reviews of new products will help you choose the best tool for the job. Columns will keep you up-to-date on C and the SDK, C++ and class libraries, database engines, OOP languages like Actor, Pascal, and Smalltalk, and tools like Visual Basic. A "first look" section will give you advance information on the latest products. We'll tell you which books to add to your Windows library and keep you informed on industry developments.

Plan for a programming breakthrough. For a **FREE PREMIER ISSUE,** call **800-234-0386** or FAX your name and address to **503-746-0071.** We'll send you a free issue and start your **no-risk** subscription. You'll get a full year—12 issues in all—of Windows tools and techniques for just **$29.95.** If you're not completely satisfied, simply write "cancel" on the invoice. The free issue is yours to keep, and you owe nothing.

Whether you're just getting started with Windows or have been developing for years, **Windows Tech Journal** is for you.

Windows Tech JOURNAL

The new magazine of tools and techniques for Windows developers. To subscribe, call 800-234-0386.

Contact this advertiser directly

49103

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

A window system for text editing

Add an editing layer to your windowing system in an object-oriented manner

By Allen Holub

It's time to wind up the windowing system that I've been presenting in recent issues (*PJ* 9.1, 9.4, and 9.5) with the lowest-level window-management functions. Last issue I discussed high-level functions that affect entire windows (create them, delete them, move them, and so forth). The low-level functions manipulate the contents of the window.

It's no accident that this specific set of low-level functions exists. These functions are the ones that a general-purpose editing engine needs to implement a window-based editor. They provide all the primitive operations that are required to manipulate the text in a window. Although many of these functions are similar to the lower-level **viewport** and **textbuf** functions, the difference is that the current functions manipulate both the **viewport** and the **textbuf** components at the same time. **e_ins,** for example, inserts a character into the text buffer and then updates the viewport to display the updated buffer. The high-level editor calls **e_ins** to insert a character, and **e_ins** takes care of the low-level details of keeping the text buffer and viewport in synch.

Another example is **e_gotorc,** which moves the cursor to a specified line and column number in the underlying file. To do so, it must do one of three things. If the desired position is in the viewport, it just puts the viewport and text-buffer cursors in the right place. If the viewport is smaller than the text buffer and the desired position is in the text buffer but not the viewport, **e_gotorc** must reposition the viewport with respect to the text buffer. Finally, if the desired position isn't in the text buffer, **e_gotorc** must scroll both the viewport and text buffer to the correct line and, if necessary, bring in new lines from (and export modified lines to) the underlying file.

Allen Holub is a teacher, writer, consultant, and composer. He teaches C and compiler design for the U.C. Berkeley Extension. His most recent book is Compiler Design in C. *Allen can be reached at holub@violet.berkeley.edu (...!ucbvax!violet!holub) via USENET or c/o Software Engineering Consultants, P.O. Box 5679, Berkeley, CA 94705.*

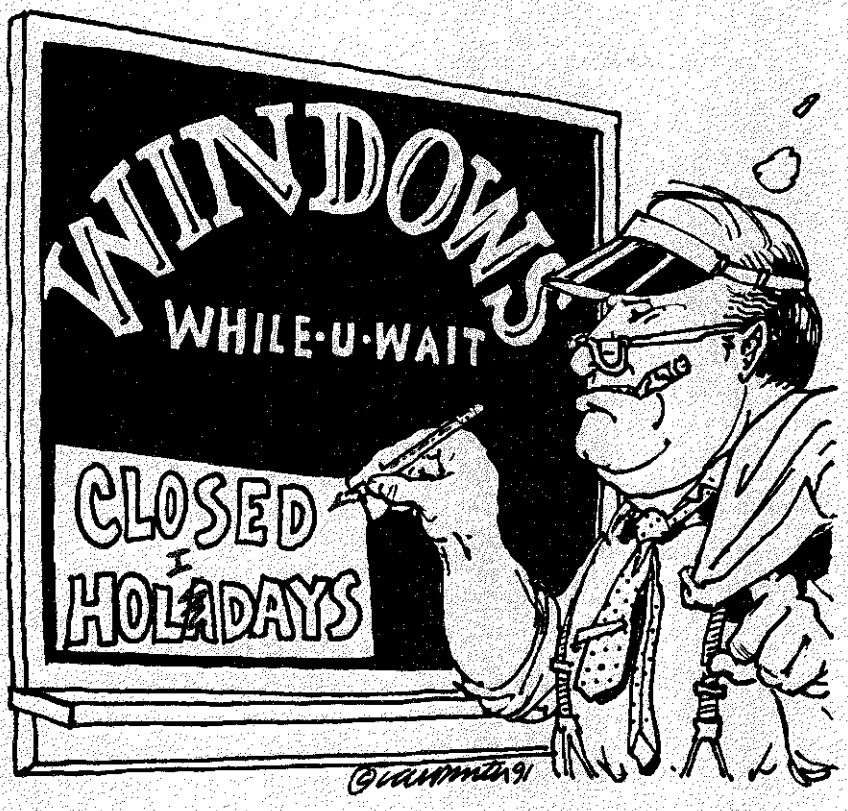
DATA ABSTRACTION

I've written the code for the entire system in as object-oriented a manner as possible. The main OOP buzzword of interest here is *data abstraction:* I've isolated the mechanics of data manipulation from the outside world, layering the various systems to make more-complex systems. It would be instructive to look at how all this could be implemented in C++, which I plan to do at some juncture.

Because the window system is built around the possibility of adding an editor at a future date, the editor is central to the whole system. The editor forms a class—a black box that's manipulated by sending it messages. Here, the messages comprise a stream of characters that are sent to the editor one at a time via a send-character-to-editor function. The editor manipulates the text in the window, but it's written so that all the details of manipulating both the text and the viewport are hidden from it. This made it possible for me to write the editor without knowing what the actual display system was going to look like. The editor modifies the display by invoking workhorse functions that are called *methods*.

The methods can be attached to the editor in one of two ways. First, the structure that holds the editor's state information (the **editor** structure from last time) can hold pointers to the various workhorse functions. When the editor needs to move the cursor, for example, it calls the move-cursor function indirectly through the pointer in the **editor** structure. The advantage of this approach is that several editors, each of which displays to a different device, can exist at the same time. The method pointers in the editors' **editor** structures would point to various subroutines, depending on the display environment. This way a program could simultaneously have an editable window on the main screen and have a window displayed on a dumb terminal connected to the machine through

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED



a serial port. The same editor code is used for both situations, but each editor would have a unique set of workhorse functions: one set for the main screen and another for the dumb terminal.

The second mechanism for attaching method functions to an editor is simpler but less flexible. The editor code simply makes direct function calls when it needs to manipulate the screen. The programmer must provide versions of those functions, and the linker attaches the functions to the editor at link time. You still have the advantage of not having to modify the actual editor code to customize it for a particular environment, but you lose the flexibility. Because there's only one set of display functions, all instances of the editor in a given program must manipulate the same display.

BEYOND THE HOOPLA

Although none of the foregoing is earth-shattering, the techniques are presented as such by OOP proponents. To cover up the fact that nothing they're talking about is really as revolutionary as claimed, the OOP pushers have come up with a slew of buzzwords to obfuscate the issue.

First of all, the editor structure is called a *class*, not a structure. A declared instance of the structure is called an *object*. The method functions manipulate objects when they are passed pointers to the specific object being manipulated. This way, one set of functions can manipulate various objects. This commonplace process is obscured when it is described as the object receiving a message rather than as a function being called by passing it a pointer to a structure.

The OOP crowd claims that a message is not just a function call. A call such as **foo(&a_struct),** they argue, is just an encoding of the real process: pass the "foo" message to the **a_struct** instance of a class. I don't think much of the argument, though. Using functions is by far the most common way to pass messages to objects—it's the only way to do it in C++, for example. I see little point in obscuring this fact with ideas about how the message could be passed in some theoretical environment.

There are also OOP buzzwords to describe the ways that the method functions are called. The first system, in which the method functions are called through pointers, is called *dynamic* or *late binding*, because the methods are dynamically connected to the editor late in the program-creation process (at run-time, not compile time). Functions called indirectly through pointers are called *virtual* functions.

Various editor objects (declared instances of the editor structure) can have different method functions attached to them, so they might interpret the same message in different ways. That is, when a function is called with pointers to two different editor structures, those instances of the editor might interpret a request to move the cursor in different ways by executing different method functions. This process is called *polymorphism* (a great buzzword—it's English, but nobody knows what it means). The second binding system, in which the method functions are bound to the editor at link time, is called *static* or *early* binding.

DOWN TO WORK

The functions described in this installment are the method functions that my implementation of an editor needs to get its work done. My own implementation uses static binding, which is a bit more efficient than dynamic binding. The editor structure is smaller because you don't need the pointers, and the calls are marginally faster because you don't have the indirection. I don't expect to need multiple editors that talk to different output devices.

The first file, EDITOR.H (Listing 1), describes the **editor** structure. A field of this type was declared at the top of last issue's **win** structure, so a pointer to a **win** is also a pointer to an **editor.**

OOP proponents have their own ways to describe the kind of relationship in which a pointer to a structure can also be used as a pointer to a nested substructure. The substructure (the **editor**) is called the *base class*, and the outer structure (the **win**) is called the *derived class*. A **win** pointer can safely be converted to an **editor** pointer using a simple cast, but the reverse isn't true. There's no guarantee that an **editor** structure is actually inside the **win** structure, so casting an **editor*** to a **win*** could be dangerous if no surrounding **win** exists. What an OOP enthusiast who says "A derived-class object can be promoted to a base-class object" means is that it's safe to cast a subclass pointer to a pointer to the enclosing structure.

Because the current implementation contains no editor, the **editor** structure (which is at the bottom of the listing) is empty. You can fill it in with whatever the editor actually needs when you write it.

Listing 1 also provides function prototypes for the remainder of the window-management functions, which are the workhorse functions needed by the editor to do its work. Three prototypes to interface with the editor are also provided:

```
int engine       (ETYP *const p, int c);
int engine_open  (ETYP *const p);
int engine_close (ETYP *const p);
```


ILLUSTRATION BY JIM CARPENTER

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Simplify Real Time Programs with

INTERWORK™

Concurrent Programming Toolkit

New version 3.1 now available

Interwork is a "C" program library that lets you write your programs as a set of cooperating concurrent tasks. Very useful for real-time applications, simulation, and parallel programming.

FEATURES

- Unlimited number of tasks
- Fast context switching
- Optional time sliced scheduling

NEW! • Enhanced inter-task communication
NEW! • Improved interrupt and trap handling
NEW! • Expanded user's guide with examples

Interwork is available for the following systems:

IBM PC, XT, AT	PC-DOS 2.0 or later	$229
IBM PC AT	XENIX 286	$279
Intel 386	UNIX V.3/386	$369
DEC VAX, Sun 3,4	UNIX 4.2BSD	$369

For more information please contact:

Block Island Technologies
Innovative Computer Software
15455 NW Greenbrier Parkway, Suite 210
Beaverton, Oregon 97006
(503) 690-7181 FAX (503) 645-7732

Circle **19** on Reader Service Card

SUBSCRIBE!

VLIB

The User Interface Library for C

Create Fast and Compact Applications in C & C++

✦ Menus ✦ Forms ✦
✦ Editor ✦ Windows ✦
✦ Mouse Support ✦
✦ Pick Lists ✦
✦ Popup Messages ✦
✦ Dialog Boxes ✦
✦ Colors ✦ and more! ✦

VLIB is a library of over 300 C functions for building PC applications. VLIB produces the *smallest, fastest* programs of *any* C library!

For *Microsoft C* and *Quick C, Borland Turbo C* and *C++*. All memory models.

Free Demo Disk Available

Call (408) 984-2256
Pathfinder Associates
291 Madrone Avenue
Santa Clara, CA 95051
FAX (408) 244-5665
BBS (408) 246-0164

VLIB $ 149
libraries, manual, and source code.

VMAN $ 49
online hypertext manual for VLIB.

Circle **20** on Reader Service Card

engine_open initializes the editing engine, initializing the **editor** structure as necessary, and **engine_close** shuts the engine down. The actual interface, **engine,** is passed a message stream one character at a time by means of successive **engine** calls.

Because **engine_open** is called indirectly by **w_create** and **engine_close** by **w_destroy,** you'll need to provide a couple of empty functions to keep the linker happy until you implement an actual editor.

To cover up the fact that nothing they're talking about is really as revolutionary as claimed, the OOP pushers have come up with a slew of buzzwords to obfuscate the issue.

The rest of EDITOR.H contains prototypes for a number of editor workhorse functions that update the screen and the underlying buffers and files. These functions are tailored to our windowing environment, but you can customize the editor to a specific environment by changing them to suit your application.

All the functions take a **p** argument, which is the same **p** that is passed to the editor—the double **editor** and **window** pointer discussed above. Note that the editor doesn't actually know anything about windows; it just relays **p** to the workhorse function.

There's one final caveat. The workhorse functions use 0 for the top row and 0 for the leftmost column index. If the editor uses 1 for the top line (as is usually the case), this can cause confusion.

THE WORKHORSE FUNCTIONS

The editing window functions are in EDIT.C (Listing 2). In the current implementation, the ETYPE macro that's used in the following prototypes is mapped to **struct win** before EDITOR.H is **#include**d in the program.

The editor needs four functions to make decisions about the viewport:

```
int e_vrow   (ETYP *const p);
int e_vcol   (ETYP *const p);
int e_vnrows (ETYP *const p);
int e_vncols (ETYP *const p);
```

The first two return the current viewport-relative row and column coordinates of the cursor (not the absolute, file-relative coordinate). The second two functions yield the number of rows and columns that can be displayed in the viewport (its height and width).

LINE EXAMINATION

The engine uses an additional set of functions to examine single lines of the underlying text buffer. From the perspective of the editor, an implementation doesn't have to have a separate buffer (as is true when it's using the screen itself as its buffer); in this case, the functions would just examine the video memory. If it does have a separate buffer, as the current implementation does, the following functions can examine the buffer without having to look at the screen at all.

The first couple of line-examination functions

```
int e_first (ETYP *const this);
int e_last  (ETYP *const this);
```

return the column index of the first and last nonwhite characters on the current line (the line the cursor is resting on). Both functions return 0 for a blank line. The more general function

```
int e_find_c( ETYP *const this, int c,
              int forward, int incl );
```

looks for the character **c** on the current line. If the character is found (to the right of the current cursor position if **forward** is true, to the left otherwise), **e_find_c** returns the column index of **c.** Otherwise, it returns –1. If **incl** is false, **e_find_c** returns the index of the character next to the **c** rather than the index of **c.** This is the character to the right of **c** if we're looking backwards, the character to the left of **c** if we're looking forward.

The final line-examination function

```
int e_cur(ETYP *const this);
```

looks at characters in the buffer. **e_cur** returns the character at the current cursor position. Note that **e_cur** is occasionally

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C CODE FOR THE PC

source code, of course

	M++ (fast C++ math classes; indexing, matrices & vectors, numerics, memory handling, I/O; specify C++ compiler/Turbo, Zortech, Glock.)	$395
	C++/Views (C++ interface to MS Windows 3.0; over 60 classes; free broswer (no source); no royalties; Zortech C++ only)	$395
	Embedded DOS (full-features, real-time, multitasking, 3.31-compatible DOS for embedded system and self-bootable installations)	$375
NEW!	Spell Time (spelling checker for incorporation into text products; no royalty; large dictionary; small and fast)	$300
	ZIP Image Processor & Victor Image Library Version 2.0 (brightness, contrast, merge images, TIFF/GIF/PCX/bin, HP ScanJet support)	$290
	TurboTEX (Release 3.0; HP, PS, dot drivers; CM fonts; LaTEX; MetaFont)	$250
	db_File & db_Retrieve Bundle by Raima (B-tree and network database with SQL query and report writer; multi-user $475)	$245
NEW!	Meijin++ (C++ class library for modeling & simulation; queues, statistical tools, fourier transforms, differential equations; no royalties)	$220
	TE Editor Developer's Kit for Windows (full screen editor, undo command, multiple windows; inc. TER for application build-in; no royalties)	$220
	c_pslib (PostScript generation library for C programs; includes complete graphics, font, rotation & paragraph support	$170
	Rogue Wave tools.h++ or math.h++ Class Library (extensive docs)	each $165
	WKS Library Version 2.01 (C program interface to Lotus 1-2-3, dBase, Supercalc 4, Quatro, & Clipper)	$155
	Minix Operating System (Version 1.5; Unix-like operating system, includes manual; specify 5.25" or 3.5" diskettes)	each $150
	Delorie GCC for MS-DOS (Version 1.39; includes C++, assembler & DOS extender; complete source code and makefiles)	$150
	CBTree (B+tree ISAM driver, multiple variable-length keys)	$135
	TE Editor Developer's Kit (full screen editor, undo command, multiple windows; with word processing: $195; Windows 3.0: $200)	$130
	C Communications Toolkit by Magna Carta (multi-port & co-processor board support, FAX, interrupt driven, emulations, xfer protocols)	$125
	C++ Object Library (virtual windows, I/O, lists, file free space management, keyed ISAM file I/O)	$125
	Booter Toolkit (floppy disk bootstrap routines, DOS file system, light-weight multitasking, windows, fast memory management)	$120
	V64 (virtual/extended memory management library, 64Mb address space, fast, direct access to any byte, checkpoint/restart from file)	$120
	WinMem (unlimited global memory handles for Windows 3.0, 4-byte overheadper block rather than 20, virtual memory for 286)	$110
	Network Toolbox/N (C functions for Novell Netware 2.1x, sample programs included)	$110
	SCM (portable Scheme in C, conforms to IEEE and 3.99 specs, garbage collection off C stack for easy mixing with C)	$100
Updated!	PC/IP (CMU/MIT TCP/IP for PCs; Clarkson drivers, SOSS NFS server, Bdale mailer, PCRoute & PCBridge, NDIS, ODI, Beholder, more)	$100
	Heapman (application memory management for Windows 3.0; 64K bytes of heap space; includes memory browser/debugger)	$100
	VolksWindows (simple graphical interface to DOS file system; copy, launch programs, move files, etc.)	$100
NEW!	EZTrieve (Novell Btrieve access with data dictionary and data manager; no royalties)	$100
	CDB for DOS & Unix (database toolkit; ISAM, DDL, relational & network, space reuse, concurrent access; fast, no royalty)	$95
NEW!	Kier FinanceLib (interest, conversions, annuities, depreciation, cash flow, bonds, etc.)	$95
	PowerSTOR (Version 1.2; extended heap space on extended memory, expanded memory, and/or hard disk)	$95
NEW!	Script Interpreter (a command script interpreter for DOS-based systems; C-like script language; lots of features)	$90
	MIRACL Multiprecision Integer & Rational Arithmetic C Library (nearly 100 functions; C & C++; from Ireland)	$90
	VMEM (virtual memory manager by Blake McBride, Version 3.6, LRU pager, dynamic swap file, image save/restore)	$80
	cbase (C database library, sequential & b+-tree random access, buffered block I/O, file-level locking)	$75
NEW!	TreeDraw (PostScript display of labeled hierarchical trees; DOS & Windows screen display included; Moen algorithm)	$75
	COMM-DRV (interrupt-driven serial communication, device drivers & serial port monitors; source for libraries only; complete source $200)	$75
	FlexList (doubly-linked lists of arbitrary data with multiple access methods; specify C or C++)	each $65
	Foundations-1 C++ Class Library (ASCII record I/O, bit arrays, exception handling, B-tree, persistent objects)	$60
	Kier DateLib (all kinds of date manipulation; translation, validation, formatting, & arithmetic)	$60
	Coder's Prolog (Version 3.0; inference engine for use with C programs)	$60
	FinanC (large collection of financial function including bond, inventory, stock portfolio, & cash flow)	$55
	CALC (ASCII algebraic expression evaluator, unlimited parenthesis nesting, symbols, 32 built-in functions, easily extended)	$50
	Mini IDL (interface generator for complex data; single inheritance, translator & table generator tools, ASCII external form only))	$50
	Backup & Restore Utility by Blake McBride (multiple volumes, file compression & encryption)	$50
	TILE Forth (32-bit Forth-83 in C; lots of extensions including multi-tasking & classes/instances;link in C routines; DOS & Unix)	$50
	Floppy TAR (TAR backup and restore on MS-DOS devices; direct access to non-standard devices)	$50
	C Compiler Pack (5 C compilers; 3 for 8086 (gcc, MicroC, Small C), 2 for 68000 (Sozobon, cc68); gcc ports include library source only)	$50
	SuperGrep (exceptionally fast, revolutionary text searching algorithm; also searches sub-directories)	$50
	OBJASM (convert .obj files to .asm files; output is MASM compatible)	$50
	YATE (Yet Another Text Editor, written in C++, full-screen)	$50
	CLIPS Version 5.0 (rule-based expert system generator; advanced manuals available at additional cost)	$50
	NIH Class Library & Book (basic C++ classes & *Data Abstraction and Object-Oriented Programming in C++* in softback by Keith Gorlen)	$50
	Editor Pack (fifteen public domain editors; including microEmacs 3.11, Stevie, Elvis, Moke, mg2a, DTE, Jove, & Origami)	$50
NEW!	NETS Version 2.01 (neural net simulator from COSMIC)	$40
	DES Encryption & Decryption (2500 bits/second on 4.77 MHz PC for on-the-fly encryption at 2400 baud; domestic distribution only)	$40
	RXC & EGREP Version 2.0 (Regular Expression Compiler and Pattern Matching; finite state machine from regular expression)	$35
	Bison & BYACC (YACC workalike parser generators; documentation; no restrictions on use of BYACC output)	$35
	Spell Pack (6 spelling programs, a hyphenator, 2 utility packs and a 60K word list: Ispell, Microsp, Sp, Cspella, Spell, Dawg, Soundex)	$30
	REGX Plus (search and replace string manipulation routines based on regular expressions)	$30
	GNU Awk & Diff for PC (both programs in one package)	$30
	Make Pack (eight versions of Make including dmake 3.7, GNU Make, Cake, and Gymake and a makefile maker)	$30
NEW!	Big Number Pack (6 arbitrary precision arithmetic packages in C, one in Fortran but free Fortran-to-C converter is included)	$30
	Crunch Pack (30 file compression & expansion programs)	$30
	GNU Ghostscript (Turbo C makefile, MS-DOS executable, Paintjet driver, all fonts)	$25
	BigNum (portable and efficient arbitrary-precision arithmetic package; hardcopy docs; from France)	$25
NEW!	Database Pack (seven databases, simple to complex: BPLUS, AVL, SDB, ID, GDBM, REQUIEM, INGRES89)	$25
	String Pack (lots of string routines; C++ String class, BAWK, word wrap, fuzzy search, Boyer-Moore, Hypertext, etc.)	$25
Updated!	UUPC Pack (UUCP for the PC; UUPC Version 1.11k by Wonderworks and smail/PC Version 2.5 by Stephen C. Trier)	$25
	PERL for MS-DOS (C, sed, awk, and shell all rolled into one language; includes hardcopy docs)	$25
	Tcl (Tool Command Language; add shell programming programming capability to any command line; elegant command line language)	$25
	FLEX (fast lexical analyzer generator; new, improved LEX; BSD Version 2.3.6 with docs)	$25
NEW!	Livermore Loops in C (the famous Fortran benchmark transliterated to C; includes technical report as PostScript file)	$20
	XLISP 2.1 (includes Almy improvements)	$20
	GNU RCS (FSF's version of the Revision Control System; like Unix's SCCS only better; keeps track of software development)	$20
	SDBM (fast, disk-based hash table manager for really large hash tables; clone of Unix *ndbm*)	$20
	Data	
	Moby Thesaurus (25K root words, 1.2M synonyms; requires signed license agreement)	$350
	Moby Part-of-Speech (200,000 words and phrases described by prioritized part(s)-of-speech)	$120
	Moby Words (500,000 words & phrases, 9,000 stars, 15,000 names)	$80
	Moby Shakespeare (plays, sonnets, etc. ... every last word)	$60
	Dictionary Word List (234,932 words in alphabetical order)	$60
	Roget's 1911 Thesaurus	$40
	U. S. Cities (names & longitude/latitude of 32,000 U.S. cities and 6,000 state boundary points)	$35
	The World Digitized (100,000 longitude/latitude of world country boundaries)	$30
NEW!	Lots 'O Words (160,086 German, 178,430 Dutch, 61,843 Norwegian, 60,453 Italian, 138,257 French, 53,142 English)	$30
	Linguistic Experiment Dictionary (73,476 offbeat words with status and part-of-speech)	$30
NEW!	Dan Klein's Dictionaries (53,091 words and phrases in 25 dictionaries)	$20

The Austin Code Works
11100 Leafwood Lane
Austin, Texas 78750-3587 USA

much more ... ask for catalog

Voice: (512) 258-0785
FAX: (512) 258-1342
E-mail: info@acw.com

Free surface shipping for cash in advance **For delivery in Texas add 7%** **MasterCard/VISA**

Circle 21 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
#ifndef __EDITOR_H
#define __EDITOR_H

#ifndef ETYP            /* If no user definition, it's void. */
#define ETYP void
#endif
                /* These are the functions that the editor requires to
                 * manipulate the screen.
                 */
void e_attention(ETYP *const p, int horiz);  /* misc. functions */
int  e_error    (ETYP *const p);
int  e_close    (ETYP *const p);
int  e_open     (ETYP *const p);

int  e_vncols   (ETYP *const p);        /* viewport functions   */
int  e_vnrows   (ETYP *const p);
int  e_vcol     (ETYP *const p);
int  e_vrow     (ETYP *const p);
int  e_draw     (ETYP *const p);

int  e_cur      (ETYP *const this);     /* line examination     */
int  e_find_c   (ETYP *const this, int c, int forward, int incl);
int  e_first    (ETYP *const this);
int  e_isword   (ETYP *const this, int c);
int  e_last     (ETYP *const this);

int  e_closeline(ETYP *const this);     /* window functions     */
void e_cleartoeol(ETYP *const this );
int  e_col      (ETYP *const this);
void e_del      (ETYP *const this);
int  e_gotorc   (ETYP *const this, int r, int c );
void e_ins      (ETYP *const this, int c);
int  e_maxcol   (ETYP *const this);
int  e_maxrow   (ETYP *const this);
int  e_openline (ETYP *const this, int below );
int  e_row      (ETYP *const this);
void e_replace  (ETYP *const this, int c);
int  e_update   (ETYP *const this, int enable );
int  e_vscroll  (ETYP *const this, int up, int import);

                        /* special row arguments to e_gotorc()   */

#define E_HOME 0x8001   /* home row (row 0 of viewport), col. 0  */
#define E_MID  0x8002   /* middle row (of viewport), column 0    */
#define E_LAST 0x8003   /* last row (of viewport), column 0      */

#define E_LINEMAX 256   /* maximum line length                   */


#define E_MARKMAX   ((26+10)-1) /* largest index into mark array */
#define E_DEFMARK    0          /* default mark, must be zero    */
#define E_MARKNUM(c)(   isdigit(c) ? ( (c)-'0'          ): \
                        isalpha(c) ? ( tolower(c)-'a'+10 ): E_DEFMARK \
                    )
#define E_BUFMAX    ((26+10)-1) /* letters + numbers + default */
#define E_BUFNUM(c) (   isdigit(c) ? ( (c)-'0'          ): \
                        isalpha(c) ? ( tolower(c)-'a'+10 ): 0 \
                    )

/* The editor structure, used by an editor to keep track of
 * local variables. In a C++ class, this would be an editor class of which
 * all the workhorse functions would be protected virtual functions. The
 * engine() itself would be a public statically bound function, and all
 * other functions in engine.c would be private member functions. engine_-
 * open() and engine_close() are the constructor and destructor.
 */
typedef struct editor
{
    /* Add data needed by editor to keep track of its state here. */
}
editor;

/* These are the interface functions to the engine. */

extern int      engine          (void *const p, int c);
extern int      engine_open     (void *const p);
extern int      engine_close    (void *const p);
#endif /* __EDITOR_H */
```

LISTING 2

```
#include <stdio.h>
#include <stdlib.h>
#include <stdarg.h>
#include <ctype.h>
#include <dos.h>
#include <conio.h>
#include <string.h>
#include <tools/viewport.h>
#include <tools/textbuf.h>

#include <tools/window.h>

/* Interface functions to test the edit engine are described in engine.h.
 * The following functions are used internally. They do not interface
 * with the edit engine.
 */
static int  draw    (win *const p, int ask_top, int ask_left );
static int  hscroll (win *const p, int left);
static int  bottom  (win *const p, int offset);
static int  del_line(win *const p, textbuf *const b,int which_line);
static int  export  (win *const p, textbuf *const b,int which_line);
static void status  (win *const p, char *fmt, ... );

/*------------------------------------------------------------------------*/
int     e_open( win *const this )
{
    /* Called when engine for a specific window is opened. Try to initial-
     * ize buffer by importing the top few lines. On the first call, the
     * import function should take care of both opening the file and adjust-
     * ing this->bottom to hold the line number of the bottom line in the
     * file. It should set up the line-number table at this time, as well.
     */
    textbuf  *b = &this->b;
    int nrows   = b_nrows(b);
    int row, i;
    char *new_line;

    if( this->import )
    {
        for( row = 0; row < nrows; ++row )
        {
            if( !(new_line = (*this->import)(this,row)) )
                break;

            bottom( this, +1 );
            b_gotorc(b, row, 0);
            for( i = b_ncols(b); --i >= 0 && *new_line ;)
                *b_advance(b) = *new_line++;
        }
        e_gotorc(this,0,0);
        draw    (this,0,0);
    }
    return 1;
}
/*------------------------------------------------------------------------*/
int     e_close( win *const this )
{
   /* Prepare for closing the window. Save the window state by exporting
    * the current buffer and various important offsets. This function may
    * be called several times.
    */
   textbuf  *b = &this->b;
   viewport *v = &this->v;
   int nrows   = b_nrows(b);
   int row;

   if( this->export )
   {
      for( row = 0; row < nrows && row <= this->bottom; ++row )
      {
          b_gotorc(b, row, 0);
          if( lexport( this, b, this->file_off + row ) )
              break;
      }
      status( this, "%%%% row, viewport = %d %%%%",v_row(v)      );
      status( this, "%%%% col, viewport = %d %%%%",v_col(v)      );
      status( this, "%%%% row, offset   = %d %%%%",this->row_off );
      status( this, "%%%% col, offset   = %d %%%%",this->col_off );
      status( this, "%%%% file offset   = %d %%%%",this->file_off);
   }
   return 1;
}
/*------------------------------------------------------------------------
 * Little stuff:
 * e_vrow       Current viewport row (top == 0)
 * e_vcol       Current viewport column (left == 0)
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

asked to examine a character to the right of the index returned by **e_last.** It should return a whitespace character (**'\n'** or space) in this situation, even if no such character is present in the actual buffer.

VIRTUAL COORDINATES

The remainder of the workhorse functions manipulate the screen, the buffer, and possibly the underlying file, all at the same time. These functions use "virtual" (file-relative) coordinates: the top line of the file is row 0, and the leftmost column is column 0. In a system that has no underlying file, the virtual coordinate is the same as the buffer coordinate. In the more complex case of a file that is larger than the buffer, you need to store a *file offset*—the distance from the top line of the file to the top line of the buffer—where these functions can get at it. That's what the **file_off** field in the **win** structure does. If the virtual coordinate is in the text buffer but not in the viewport, then the workhorse function must slide the text buffer under the viewport until the cursor position is visible. If the virtual coordinate is for a line that's in the underlying file but not in the buffer, the buffer must be reloaded from the underlying file.

WINDOW WORKHORSES

The first group of window functions is used to examine the window:

```
int e_maxrow    (ETYP *const this);
int e_maxcol    (ETYP *const this);
int e_row       (ETYP *const this);
int e_col       (ETYP *const this);
```

e_maxrow returns the largest legal row number (the bottom line in the file). **e_maxcol** holds the largest legal column index, which can be larger than the viewport width if an underlying buffer is supported, as is the case here. **e_row** and **e_col** return the current virtual coordinate for the cursor—they return the file-relative line number of the row on which the cursor is resting (not the viewport coordinate) and the distance from the leftmost character on the line (not from the left edge of the viewport).

The next function

```
int e_update( win *const this, int enable )
```

is used to disable updating when the window is hidden. It takes care of situations where the editor might need to examine the entire file (rather than just one line of the buffer) without wanting to change the contents of the viewport while it's looking around. Such an examination could conceivably modify the buffer (if a file line that's not in the buffer needs to be examined, for example), and it could move the virtual cursor anywhere in the file. Because it would be confusing for the viewport to change during the examination process, **e_update** is called to disable viewport updating while the examination is going on.

Updating is disabled with an **e_update(p,0)** call, which takes a snapshot of the current relative positions of the viewport, buffer, and file, and then disables further updating. A call to **e_update(p,1)** puts the buffer, file, and viewport back in the positions they occupied before updating was disabled, and it also re-enables updating.

e_update returns true when it changes state successfully. It returns false if it was already in the required state. You

COMPUTER LANGUAGE PRODUCTIVITY AWARD 1990

PC-lint 5.0 presents C Bug # 648

```
#include <stdio.h>

int main ()
    {
    int n = 400 * 400 / 400 ;
    printf ( "%d\n" , n ) ;
    return 0;
    }
```

What is printed? It looks like it should be 400 but if you try this on MS-DOS, you will get 72. Where's the error? Compilers won't tell you.

PC-lint will catch this and many other C bugs. Unlike your compiler, PC-lint **looks across all modules** of your application for bugs and inconsistencies.

New – Optional Strong Type Checking and variables possibly not initialized.

More than **330 error messages.** More than **105 options** for complete customization. Suppress error messages, locally or globally, by symbol name, by message number, by filename, etc. Check for **portability** problems. Alter size of scalars. Adjust format of error messages. Automatically generate ANSI prototypes for your K&R functions.

Attn: Power users with huge programs.

PC-lint 386 uses DOS Extender Technology to access the full storage and flat model speed of your 386. Now fully compatible with Windows 3.0 and DOS 5.0

PC-lint 386 – $239
PC-lint DOS or OS/2 – $139

Mainframe & Mini Programmers

FlexeLint ***in shrouded source form, is available for Unix, OS-9, VAX/VMS, QNX, IBM VM/MVS, etc. Requires only K&R C to compile but supports ANSI. Pricing starts at $798. Call for details.***

Gimpel Software

3207 Hogarth Lane, Collegeville, PA 19426

CALL TODAY (215) 584-4261 Or FAX (215) 584-4266

30 Day Money-back Guarantee.

PA add 6% sales tax.

PC-lint and FlexeLint are trademarks of Gimpel Software

Contact this advertiser directly

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
 * e_vnrows      Number of rows in viewport
 * e_vncols      Number of columns in viewport
 * e_isword      True if c argument is a "word"
 * e_maxrow      Largest virtual (file) row index
 * e_maxcol      Largest virtual (file and buffer) column index
 * e_col         Current virtual (file and buffer) cursor column
 * e_row         Current virtual (file) cursor row
 * e_cur         Character at current cursor position
 * e_last        Index of rightmost nonwhite character on line,
 *               zero if the line is blank
 */
#pragma warn -par /* Turn off warnings about arguments not used. */

int  e_vrow   (win *const w) { return v_row     (&w->v);  }
int  e_vcol   (win *const w) { return v_col     (&w->v);  }
int  e_vnrows (win *const w) { return v_nrows   (&w->v);  }
int  e_vncols (win *const w) { return v_ncols   (&w->v);  }
int  e_isword (win *const w,
                    int c) { return( isalnum(c) || c=='_' );}
int  e_maxrow (win *const w) { return w->bottom;          }
int  e_maxcol (win *const w) { return b_ncols(&w->b) -1;  }
int  e_col    (win *const w) { return b_col(&w->b);       }
int  e_row    (win *const w) { return b_row(&w->b)
                                    + w->file_off;}
int  e_cur    (win *const w) { return *b_current(&w->b);  }
int  e_last   (win *const w) { return b_last(&w->b);      }

#pragma warn +par
/*--------------------------------------------------------------------
 * e_attention() and e_error();
 *
 * e_attention() rings the bell if not a horizontal motion. In
 *               any event it sets the window's error flag.
 * e_error()     returns the value of the flag and resets the
 *               flag to zero.
 */
void e_attention(win *const this, int h)
{
    if(!h)
        putchar('\a');
    this->error = 1;
}

int     e_error( win *const this )
{
    int rv = this->error;
    this->error = 0;
    return rv;
}
/*--------------------------------------------------------------------*/
int e_first     (win *const this)
{
    // Return the column index of the first nonwhite character
    // on the current line or 0 if the line is blank.

    textbuf *const b  = &this->b;
    int     start_col = b_col(b);
    int     start_row = b_row(b);
    int     last_col  = b_ncols(b) - 1;
    int     col;

    b_gotorc(b, start_row, 0);
    for( col = 0; (col<last_col) && isspace(*b_current(b)); )
    {
        ++col;
        b_advance(b);
    }
    if( isspace(*b_current(b)) ) /* Line was entirely blank. */
```


Program FASTER with ...

CC-RIDER

or the new

C++ RIDER

POP-UP SOURCE BROWSERS FOR ANY EDITOR!

CROSS-REF Editing!
Instantly pop-up your symbol definitions with a hypertext hot-key inside any editor! The Standard Edition has Source View, Edit and Paste commands. And the powerful *Professional Edition* even lets you walk through all uses of a symbol in your code as you edit!

Source Analysis!
Build a database of symbol information with our powerful ***source code analyzer*** which generates a wealth of useful output, like fully commented static and global function prototype #include files and a detailed log file showing WHERE and HOW all your symbols are used. And it uses expanded memory for ***lightning fast*** source code analysis!

QuickHelp, too!
CCSYM can also create a ***help database for your program*** which is compatible with Microsoft's help system, *QuickHelp*. All symbols in your application are accessible from QuickHelp's menus, with ***documentation extracted automatically from your original source code.***

Works with ANSI/Microsoft C, plus AT&T, Glockenspiel, Zortech, Borland C++. Supports Microsoft C v6 extensions.

FREE DEMO !

	Standard	*Professional*
	$89	***$279***
	DOS or OS/2 C or C++	DOS & OS/2 C & C++

Western Wares
(303) 327-4898
Box C Norwood, CO 81423

Circle 22 on Reader Service Card

YOU OUGHT TO BE IN ☆ REPRINTS ☆

You're the picture of success.
And **PJ** reprints are a great way to showcase your four-star performance. Reprints help you:

- ☆ Enhance your trade show presentations
- ☆ Promote your products and services
- ☆ Increase your marketplace presence
- ☆ Optimize your product training sessions

PROGRAMMER'S JOURNAL offers full-color or black-and-white reprints.

PJ reprints keep you in the spotlight.

Call us today at 800-234-0386.

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

can use this mechanism to make sure that the function that turned off updating is also the function that turns it back on again:

```
foo(void *p)
{
    int i_disabled_updating = e_update(p,0);
    /* examine the file */
    if( i_disabled_updating )
        e_update(p,1);
}
```

e_update can also be used to suppress screen updating while a window is hidden. Note that **e_update** attempts to restore the original screen state as part of the enabling process. That is, it attempts to get the relative positions of the file, buffer, and viewport back to their original condition, and also to put the cursor in the position that it occupied when updating was disabled. If the underlying buffer was modified significantly while updating was disabled, the results of the enable can be surprising.

The next three functions insert and delete characters to and from the window (from the viewport, text buffer, and underlying file—all at once):

```
void e_ins        (ETYP *const p,int c);
void e_replace    (ETYP *const p,int c);
void e_del        (ETYP *const p);
void e_cleartoeol(ETYP *const p);
```

e_ins inserts a character (**c**) at the current cursor position, first moving the character at the cursor position and all characters to its right over one notch to make room. The character at the far right of the line is lost. (Remember that the right edge of the line might be much farther away than the right edge of the viewport). **e_replace** overwrites the current character without inserting. Both **e_replace** and **e_ins** adjust the bottom-of-file marker that's used by **e_maxrow** to reference the current line if the cursor is positioned below the last line of the file. An ANSI NUL (**c==0**) doesn't print, but the bottom-of-file marker is still adjusted if necessary.

Finally, **e_cleartoeol** replaces the character under the cursor and all the characters to its right with spaces. Note that an underlying buffer that uses variable-length strings rather than space-padded, worst-case-sized arrays does not have to fill out the string with spaces, provided that the buffer-management functions are smart enough to translate a reference to a character that's off the end of the string as a space character. (In fact, a variable-length-buffer system could make the string smaller in response to a **e_cleartoeol**.)

FILE-MOVING FUNCTIONS

None of the above window functions has had to do any actual file I/O; they've all modified things at the current cursor position, which is in the text buffer component of the window. The last four window functions, however, require file manipulation. The first two

```
int e_closeline (ETYP *const p);
int e_openline  (ETYP *const p, int below);
```

delete an existing line from the file or open up a new line in the file. **e_closeline** deletes the current line by moving the following lines up one notch and redrawing the screen. The cursor position does not change. This function indirectly calls the "delete" function (a pointer to which is included in last issue's **win** structure) to tell the file manager to delete the current line.

e_openline does the opposite: it adds a new line to the file. If **below** is true, everything below the current cursor line is moved down; otherwise, the current cursor line and everything below it are moved down. The newly added line should be blank, and the cursor should rest at its far left. This function first calls the file manager's "export" function to export the bottom buffer line to the file. This way the buffer can be scrolled down, and any changes that have been made to the bottom line are saved. The function should then call the "insert" function to tell the file manager to insert a line at the proper position.

Scrolling is accomplished by calling

```
int e_vscroll(ETYP *const p, int up,
              int import);
```

which scrolls vertically by one line, up if **up** is positive, down if it's negative. It returns false if it can't scroll because the buffer's banged up against the start or end of the file (if there is an underlying file) or if the viewport is banged up against the top or bottom of the buffer (if there's no underlying file). The scrolling opens up a line at either end of the viewport, which is filled with the appropriate text. If there is an underlying file, a scroll up, for example, would export the top line of the buffer with a call to the file manager's "export" function, shift

Need more RAM?

Just relink your C program with MoreHeap and add up to 348K of heap instantly!

What is MoreHeap?

MoreHeap is an integrated
- **far heap manager,**
- **virtual memory manager, and**
- **heap debugger**

Heap Manager

- **Up to 348K of additional heap**
- **Minimizes heap fragmentation**
- **Impact on executable size is minimal (500 bytes to 8K)**
- **As fast, or faster, than your RTL routines**
- **Automatic HIMEM, QEMM, EMM386, and DOS 5 recognition.**

Replaces far heap memory manager. No reprogramming—just relink. Malloc( ) will search out additional memory between 640K and 1MB and provide it to your program completely transparently.

Virtual Memory Manager

- **Up to 128MB of virtual memory.**
- **LRU paging algorithm minimizes swapping.**
- **As low as 1 byte overhead of addressable memory per 2K of allocated virtual memory. Great for applications requiring lots of small allocations.**
- **Swappable, discardable, and movable memory blocks.**
- **Overflow to expanded memory, extended memory and disk.**

348K not enough? MoreHeap's virtual memory manager accesses up to 128MB of virtual memory. Use malloc-like functions to allocate and free virtual memory.

Heap Debugger

Debuggers can't find these errors but our heap debug functions track them down fast.

- **Unfreed blocks.**
- **Blocks freed twice.**
- **Modifications to write protected blocks (386 not required!).**
- **Buffer overwrites.**
- **Invalid pointer usage.**

Change one line of source and recompile with a command line switch. Debugs both far heap and virtual memory. Provides detailed reports on heap transactions and reports errors found including file name and line number where error was discovered.

Royalty Free—$135—Source included.

Compatible with Turbo C 2.0 and Turbo C++ 1.0, Borland C++ 2.0, Microsoft C 5.1 and 6.0, and Aztec C 4.1. Site licenses available.

SEABREEZE SOFTWARE SYSTEMS

Voice 609 924 6793
BBS 609 683 7701
CIS 72330, 705

908 Route 518
Skillman, NJ 08558

Shipping and handling via UPS ground USA $5, Canada/Mexico $10, other countries $15. 5% cash discount. Visa/MasterCard/COD. Call for PO's and overnight shipping.

Circle **23** on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        col = 0;
    b_gotorc( b, start_row, start_col );
    return col;
}
/*---------------------------------------------------------------------*/
int e_find_c(win *const this, int c, int forward, int incl)
{
    /* If the character is found on the current line (to the right of the
     * current cursor position if "forward" is true, to the left otherwise),
     * return the column index of c. Otherwise return -1. If "incl" is
     * false, return the index of the character next to the c rather than
     * the index of c itself.
     */
    textbuf *b    = &this->b;
    int row       = b_row(b);
    int col       = b_col(b);
    int last      = b_last(b);
    int start_col = col;
    int rv        = -1;

    if( forward ) forward = 1;
    else          forward = -1;

    col += forward;  /* Start search at least one column     */
    if( !incl )      /* away if not inclusive, add one more. */
        col += forward;

    if( !(0 <= col && col <= last) ) /* Col isn't on the line. */
        rv = -1;
    else
    {
        b_gotorc(b, row, col);
        while( *b_current(b) != c   &&   !b_ateol(b) )
        {
            col += forward;
            b_gotorc(b, row, col);
        }
        rv = (*b_current(b) == c) ? col : -1 ;

        if(rv != -1  &&  !incl)        /* If not including,  */
            if( (rv -= forward) < 0 )  /* back up a notch.   */
                rv = -1;

        b_gotorc(b, row, start_col);
    }
    return rv;
}
/*---------------------------------------------------------------------*/
static void status( win *const this, char *fmt, ... )
{
    /* Works like fprintf, but outputs string using the current window's
     * export function. String is exported as line -1.
     */
    char scratch[W_LINEMAX];
    va_list args;
    int len;
    if( this->export )
    {
        va_start(args, fmt);
        len = vsprintf(scratch, fmt, args);
        va_end(args);
        (*this->export)( this, -1, scratch, len+1, len );
    }
}
/*---------------------------------------------------------------------*/
static int export( win *const this,textbuf *const b,int which_line)
{
    /* Export current buffer-cursor line as line which_line.
     * Cursor does not move.
     */
    int row  = b_row(b);
    int col  = b_col(b);

    char scratch[W_LINEMAX];
    char *p;
    int  i;

    if( this->export )
    {
        b_gotorc(b,row,0);
        for(p=scratch, i=b_ncols(b); --i >= 0; )
            *p++ = *b_advance(b);

        b_gotorc(b,row,col);
        (*this->export)( this, which_line, scratch, b_ncols(b),
                                                    b_last (b)+1);
        return 1;
    }
    return 0;
}
/*---------------------------------------------------------------------*/
static int del_line( win *const this, textbuf *const b,
                                          int which_line )
{
    /* Export current buffer-cursor line for delete. Cursor
     * does not move.
     */
    int row  = b_row(b);
    int col  = b_col(b);

    char  scratch[W_LINEMAX];
    char  *p;
    int   i;

    if( this->remove )
    {
        b_gotorc(b,row,0);
        for(p=scratch, i=b_ncols(b); --i >= 0; )
            *p++ = *b_advance(b);

        b_gotorc(b,row,col);
        (*this->remove)( this, which_line, scratch,
                               b_ncols(b), b_last(b)+1);
        return 1;
    }
    return 0;
}
/*---------------------------------------------------------------------*/
int e_vscroll( win *const this, int up, int import )
{
    /* Scroll text up one line if up > 0, down otherwise. If import is
     * true, the scroll won't happen unless a new top or bottom line can be
     * imported from the overflow file. If it's false, the scroll always
     * happens, no attempt is made to do an import, and the bottom (or top)
     * line will be filled with blanks. Return true on success, false
     * otherwise.
     */
    viewport *const v = &this->v ;
    textbuf  *const b = &this->b ;
    int          rv   = 1;          /* return value         */
    int          brow = b_row(b);   /* buffer row/column    */
    int          bcol = b_col(b);
    int          vrow = v_row(v);   /* viewport row/column  */
    int          vcol = v_col(v);
    int          i;
    char         *new_line;

    if( up > 0 )        /* Move text up, viewport moves down */
    {                   * relative to buffer.
                        */
        if( (this->row_off + v_nrows(v)) < b_nrows(b) )
        {
            /* Viewport isn't banged up against bottom of buffer. Slide
             * viewport over text, but don't have to modify buffer. Posi-
             * tion buffer cursor at left edge of line with which to fill
             * the viewport's opened-up bottom line: the buffer line corres-
             * ponding to the one just below the viewport. Slide the
             * viewport by incrementing row_off.
             */
            b_gotorc(b, this->row_off+v_nrows(v), this->col_off);
            ++this->row_off;
        }
        else if( !import )              /* Import is suppressed. */
        {
            new_line = "";      /* Pretend we got an empty line. */
            goto got_bot;
        }
        else if( !this->import )  /* Try to import bottom line.  */
        {
            rv = 0;                     /* no import function   */
            goto exit;
        }
        else if( (i = this->file_off + b_nrows(b)) > this->bottom )
        {
            rv = 0;                     /* Line doesn't exist.  */
            goto exit;
        }
        else if( !(new_line = (*this->import)(this,i)) )
        {
            rv = 0;                     /* Couldn't import it.  */
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

all the lines in the buffer up one notch, and then call the manager's "import" function to get the text for the new bottom line. This line is then transferred to the buffer and viewport as required.

If the **import** argument to **vscroll** is false, **e_vscroll** shouldn't import a new line; rather, it should fill the new line with blanks. This mechanism is used by **e_openline** to add a new line to the bottom of the file. It scrolls up without an import and then calls the file manager's "insert" function.

The cursor should remain under the original character: if you're scrolling the text up, the cursor position goes up as well.

The final window-level function

```
int e_gotorc  (ETYP *const this, int row,
               int col );
```

moves the cursor to the indicated virtual row and column. If the target cursor position is in both the buffer and the viewport, only the cursor moves. If it's in the buffer but not in the viewport, the viewport is repositioned over the buffer so that the cursor position is displayed. Finally, if the target position is in the file but not in the buffer, the buffer should be reloaded to hold the required line.

You can do the reloading either by scrolling or by a mass reload of the buffer in the background (that is, "export" the entire buffer, then "import" a new one), followed by a single viewport refresh. Scrolling looks nicer if the distance being traveled is relatively short. For larger distances, a mass reload works much faster. The current implementation only scrolls, so if you're moving a long distance, it will be slow.

Note that there's no horizontal-scrolling workhorse function, because the engine never needs to do a horizontal scroll directly. If the required cursor position is not in the viewport, the engine calls **e_gotorc,** which must do a horizontal scroll. (The target position will always be in the buffer if you define the buffer width as being the same as the maximum line width in the file.) **e_gotorc** can call **e_vscroll** to do any vertical scrolling.

Three special values for the **row** argument to **e_gotorc** are defined in EDITOR.H:

```
E_HOME Go to left edge of home (top) row of
       viewport.
E_MID  Go to left edge of middle row of viewport.
E_LAST Go to left edge of last row of viewport.
```

MISCELLANEOUS FUNCTIONS

Several miscellaneous functions are also necessary. The first

```
int e_isword(ETYP *const p, int c);
```

returns true if **c** is a legitimate character in a "word." This function lets you change the definition of "word" on an editor-by-editor basis and can be handy when several editors are active at once.

The next two calls are

```
int e_open ( ETYP *const p );
int e_close( ETYP *const p );
```

e_open prepares the window for opening by importing the first buffer from the file. **e_close** prepares for closing by exporting

Building a new computer, or upgrading an old one?

Start with an AMI motherboard.

AMI, the leading manufacturer of BIOS, has developed a full line of motherboards. Since these are built with AMI BIOS, compatibility is never a problem. And neither is performance — AMI motherboards are made with the same quality discipline that has made AMI a respected industry leader.

So whether you have an old computer that needs upgrading or you just like building computers, the way to begin is with an AMI motherboard. We stock a variety of them:

ISA 386SX (AT or XT footprint)
ISA 386/486 (AT or XT footprint)
EISA (AT footprint)

Send for catalog and price list.

BCM, Dept 24
49 Sherwood Terrace, Suite R
Lake Bluff, Illinois 60044
TEL: 708-234-5576
FAX: 708-234-2522

Circle 24 on Reader Service Card

People are talking about us.

F77L-EM/32 & Lahey Ergo OS/386

Port mainframe programs as large as 96MB to 386/486's with this 32-bit DOS-Extender compiler. The Winner of *PC Magazine's* 1988 Technical Excellence Award just got better. New Version 4.0 includes: Programming Tools, Popular Fortran 90 features, Virtual Memory Support, DESQview Support, New Documentation and Free Unlimited Runtime Licenses. F77L-EM/32 $995 OS/386 $395

F77L

The fastest real-mode compiler available. F77L can take advantage of your 386 PC by generating 32-bit instructions. New Version 5.0 includes: Fortran 90 features, Weitek support, and Video Graphics. $595

Lahey Personal Fortran 77

Version 3.0: Full ANSI 77, Editor, Debugger, Linker, Library Manager, Microsoft and Borland C interfaces. A great learning tool at an unbeatable price. $99

Contact us to discuss our products and your needs. **(800) 548-4778**
Lahey Computer Systems, Inc. P.O. Box 6091, Incline Village, NV 89450
Tel: (702) 831-2500 FAX: (702) 831-8123

FORTRAN IS OUR FORTE

Circle 25 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
            goto exit;
        }
    else /* Got the line, add it to the buffer & viewport.  */
        {
got_bot:
        b_gotorc(b, 0, 0);          /* Export the top line. */
        export  (this, b, this->file_off );

        ++this->file_off ;
        b_closedown (b, ' ');    /* Delete top line and move */
                                 * following text up.
                                 */
        b_gotorc(b, b_nrows(b)-1, 0); /* buffer=new last line */
        for( i = b_ncols(b); --i >= 0 && *new_line ;)
            *b_advance(b) = *new_line++;
        b_gotorc(b, b_nrows(b)-1, 0); /* cursor to line start  */
        --brow;         /* Make cursor track previous position. */
        }

    /* Scroll the viewport text up one line, then copy the associated
     * text for the bottom line from the buffer to the viewport.
     */
    if( !this->update )
        {
        if( vrow == 0 )  /* Old line is gone. */
            {
            /* col of new top line */
            b_gotorc(b, this->row_off, bcol);
            }
        else
            b_gotorc(b, brow,   bcol );
        }
    else
        {
        v_scroll(v, 1, 0);              /* Scroll up one line.  */
        v_gotorc(v, v_nrows(v)-1, 0); /* last line, column 0  */

        /* Copy buffer to screen. */
        for(i = v_ncols(v); --i >= 0 ;)
            v_putc(v, *b_advance(b), 1 );

        if( vrow == 0 )     /* Old line is gone.     */
        {                   /* Can't restore cursor. */
            b_gotorc(b, this->row_off, bcol);
            v_gotorc(v, 0, vcol);
        }
        else /* Restore original cursor position. */
        {
            b_gotorc(b, brow,   bcol );
            v_gotorc(v, vrow-1, vcol );
        }
    }
}
else        /* Move text down, viewport moves up  */
{                       /* relative to buffer. */
    if( this->row_off > 0 )
    {
        /* Viewport isn't banged up against top of buffer. Slide view-
         * port over text.  Position buffer cursor at left edge of line
         * with which to fill the viewport's opened-up top line. Note
         * that b_gotorc() has side effects so the -- must be done
         * separately.
         */
        --this->row_off;
        b_gotorc(b, this->row_off, this->col_off );
    }
    else if( this->file_off <= 0 )
    {
        rv = 0;     /* May not scroll past initial top line. */
        goto exit;
    }
    else if( !import )              /* Import is suppressed. */
    {
        new_line = "";      /* Pretend we got an empty line. */
        goto got_top;
    }
    else if(!this->import )  /* Try to import new top line.  */
    {
        rv = 0;                         /* no import function */
        goto exit;
    }
    else if( this->file_off <= 0 )
    {
        rv = 0;             /* Line doesn't exist. */
        goto exit;
        }
    else if(!(new_line=(*this->import)(this,this->file_off-1)))
    {
        rv = 0;             /* Couldn't import new line. */
        goto exit;
    }
    else    /* Got the line, add it to the buffer & viewport. */
    {
got_top:
        --this->file_off;
        b_gotorc  (b, b_nrows(b)-1, 0); /* Export bottom line. */
        export    (this, b, this->file_off + b_nrows(b) );
        b_closeup (b, ' ' );            /* Delete bottom line. */

        b_gotorc(b, 0, 0);              /* buffer = new top line */
        for( i = b_ncols(b); --i >= 0 && *new_line ;)
            *b_advance(b) = *new_line++;
        b_gotorc(b, 0, 0);      /* Cursor to start of new line.  */
        ++brow;                 /* Cursor tracks previous position. */
    }

    /* Scroll the viewport text down one line, then copy the associated
     * text for the top line from the buffer to the viewport.
     */
    if( !this->update )
    {
        if(vrow >= v_nrows(v)-1)  /* Old line is gone. */
            b_gotorc(b, this->row_off + v_nrows(v)-1, bcol);
        else
            b_gotorc(b, brow, bcol );
    }
    else
    {
        v_scroll(v, -1, 0);             /* Scroll down 1 line. */
        v_gotorc(v,  0, 0);             /* first line, col. 0  */
        for(i = v_ncols(v); --i >= 0 ;) /* screen = buffer     */
            v_putc(v, *b_advance(b), 1 );

        if(vrow >= v_nrows(v)-1)  /* Old line is gone.         */
        {
            /* Can't restore old cursor position because the line's
             * not there anymore. Go to the original column, but of the
             * new bottom line.
             */
            b_gotorc(b, this->row_off + v_nrows(v)-1, bcol);
            v_gotorc(v, v_nrows(v)-1, vcol);
        }
        else      /* Otherwise restore original cursor to be under */
        {         * original character.
                  */
            b_gotorc(b, brow,   bcol );
            v_gotorc(v, vrow+1, vcol );
        }
    }
}
exit:
    monitor(this, UPDATE);
    return rv;
}
/*-------------------------------------------------------------------*/
int draw( win *const this, int ask_top, int ask_left )
{
    /* Fill the viewport with text from the buffer. The viewport is at
     * offset "top" from the top line of the buffer and "left" from the
     * left edge of the buffer. Cursor doesn't move with respect to the
     * viewport. Top and left are truncated if necessary to prevent the
     * viewport from moving off the edge of the buffer. Return false if the
     * coordinates were modified to draw things, true otherwise.
     */
    viewport *const v = &this->v;
    textbuf  *const b = &this->b;
    int col       = 0;
    int row       = 0;
    int nrows     = v_nrows( v );
    int ncols     = v_ncols( v );
    const int left = max(0, min(ask_left, b_ncols(b)-v_ncols(v)));
    const int top  = max(0, min(ask_top,  b_nrows(b)-v_nrows(v)));
    int cur_top   = top;
    int start_row = v_row(v);
    int start_col = v_col(v);

    if( !this->update ) /* No updating, pretend we succeeded. */
        return 1;

    while( --nrows >= 0 )
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

all internally buffered lines along with the following status lines:

```
%% row, viewport = viewport cursor row  %%
%% col, viewport = viewport cursor column  %%
%% row, offset   = distance from top of
                   buffer to viewport  %%
%% col, offset   = distance from left of
                   buffer to viewport  %%
%% file offset   = distance from line 0 in file
                   to top line of buffer  %%
```

The status lines are sent to the export function with a line number of –1. If **e_close** is not called as part of a "write" operation, the currently buffered lines will not be exported to the temporary file.

The final two miscellaneous functions are

```
int e_attention (ETYP *const p, int horiz);
int e_error     (ETYP *const p);
```

The first function should get the user's attention, usually by ringing the bell or flashing the screen. This function is called whenever a command cannot be performed (as when you try to issue an **h** from the leftmost column). It's not an error (and **e_attention** is not called) if a defined fallback position, such as vertical motion to a shorter line, is taken, but the cursor had to be moved left to rest under a valid character. The function also sets to a true condition an internal error flag that is maintained for each instance of an editor. You can access the error flag by calling **e_error**, which returns the current state of the flag and then resets the flag to false.

CUT AND WRAP

That's it. This powerful windowing system potentially lets you manipulate huge text buffers stored in files through a tiny viewport on the screen, and it provides support for an editor application if you want one.

There's still a little room for improvement, though. Error recovery in the high-level window functions could be improved a bit. You could also choose to expand the window concept considerably, implementing such things as boxed windows and subwindows (when you move a window, all the subwindows move with it, which is useful if you're using small windows to implement fields in a larger screen).

Finally, I haven't presented either the file layer or the editor engine. The former is pretty easy to implement—you need to provide only four functions. The editor engine is another matter. My version is a vi clone whose implementation takes about 2,000 lines of code—small if you consider what it does, but a bit big for a magazine article. A much simpler editor is, of course, possible and is much less work. Even with these chunks missing, I hope you find the functions spiffy enough to be helpful.

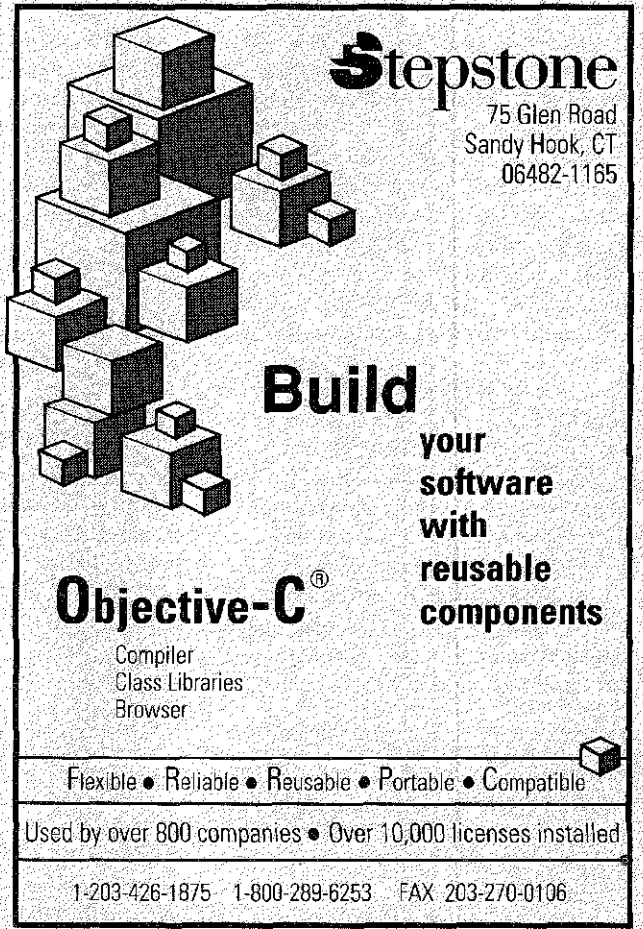
```
    {
        /* Both b_gotorc and v_gotorc have side effects. */

        b_gotorc( b, cur_top, left ); ++cur_top;
        v_gotorc( v, row,     0    ); ++row;

        for( col = ncols; --col> 0 ;)     /* all but last char. */
            v_putc(v, *b_advance(b), 1 );
        v_putc(v, *b_current(b), 0 );     /* last character     */
    }

    this->row_off = top;
    this->col_off = left;
    v_gotorc(v, start_row,       start_col       );
    b_gotorc(b, start_row + top, start_col + left );

    monitor(this, UPDATE);
    return( top == ask_top && left == ask_left );
}
/*-------------------------------------------------------------------------*/
static int hscroll( win *const this, int left )
{
    /* Scroll horizontally, "left" columns to the left (right if negative).
     * There's no import/export feature here, so a scroll past the edge of
     * the buffer is illegal. Return true if you moved the required number
     * of columns, false if not. Note that there still might have been
     * motion in the earlier case, just not enough of it. A request of zero
     * columns to the left just does a redraw.
     */
    return draw( this, this->row_off, this->col_off + left);
}
/*-------------------------------------------------------------------------*/
int e_update( win *const this, int enable )
{
    /* If "enable" is false, store a snapshot of the current viewport and
     * file states. If it's true, attempt to restore previous state (cursor
     * position and relative positions of the file, buffer, and viewport).
     * This routine is meant primarily to disable screen updating while the
     * file is being examined. Return 1 if you change state 1 if the cur-
     * rent state is requested.
     */
    if( this->update == enable )        /* nothing to do */
        return 0;

    if( !enable )                   /* disabling */
    {
        this->update = 0;
        this->update_row_off    = this->row_off;
        this->update_col_off    = this->col_off;
        this->update_file_off   = this->file_off;
        this->update_row        = e_row(this);
        this->update_col        = e_col(this);
        this->dirty             = 0;
    }
    else                                /* enabling */
    {
        /* Get the cursor back to the original place with an e_gotorc(),
         * then scroll the screen as necessary to get the cursor positioned
         * correctly relative to the viewport.
         */
        e_gotorc( this, this->update_row, this->update_col );

        while( this->file_off < this->update_file_off )
            if( !e_vscroll(this, 1, 1) )
                break;
        while( this->file_off > this->update_file_off )
            if( !e_vscroll(this, 0, 1) )
                break;

        while( this->row_off < this->update_row_off )
            if( !e_vscroll(this, 1, 1) )
                break;
        while( this->row_off > this->update_row_off )
            if( !e_vscroll(this, 0, 1) )
                break;

        this->update = 1;
        if( this->col_off != this->update_col_off )
            fprintf(stderr,"INTERNAL WINDOW ERROR: "
                           "unexpected horiz motion\n");

        if( this->dirty )
            draw( this, this->row_off, this->col_off );
    }
    monitor(this, UPDATE);
    return 1;
}
/*-------------------------------------------------------------------------*/
int e_gotorc( win *const this, int dst_r, int dst_c )
{
    /* Go to the indicated virtual row and column. If r and c are inside
     * the viewport, just go there. If not, scroll the viewport as appropri-
     * ate. New lines are imported if you request a row address that is not
     * in the buffer. Note that this subroutine might be used to add lines
     * to the end of the file (by going down from the last line).
     */
    viewport *const v = &this->v;
    textbuf  *const b = &this->b;
    int   rv = 1;                   /* return value */
    int   r, c;
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    int   nrows = v_nrows( v );
    int   ncols = v_ncols( v );

    switch( dst_r )
    {
    case E_HOME: dst_r = this->row_off;           break;
    case E_MID:  dst_r = this->row_off + nrows/2; break;
    case E_LAST: dst_r = this->row_off + nrows-1; break;
    }
    r = dst_r;

    if( !(0 <= dst_c && dst_c < b_ncols(b)) || r < 0 )
    {
        rv = 0;         // Illegal address
        goto exit;
    }
    c = dst_c;

    /* Get the desired column into the viewport, starting with column index:
     *     Modify c to hold the distance that c is from the right of the
     *     viewport. For example, c will be 0 if it's the first column past
     *     the one in the right edge of the viewport. If c is -1, it's at
     *     the right edge of the viewport. If it's -ncols then it's at the
     *     left edge of the viewport. Then, scroll as necessary to get c
     *     into the viewport.
     */
    c -= this->col_off + ncols;
    if( c >= 0 )
    {
        while( c-- >= 0 )
            if( !hscroll(this, 1) )     /* scroll left */
            {
                rv = 0;
                goto exit;
            }
    }
    else if( c < -ncols )
    {
        c += ncols;     /* c = -1 ==> 1 col to left of viewport */
        while( ++c <= 0 )
            if( !hscroll(this, -1) )    /* scroll right */
            {
                rv = 0;
                goto exit;
            }
    }

    /* Do the same thing with the vertical coordinate. The situation is
     * more complicated here, though, because the row coordinate is file
     * relative. In other words:
     *
     * r == file_off + row_off + v_row(v)
     *
     * After the subtract, below, r will hold the distance between the top
     * of the viewport and the desired row, negative if the desired row is
     * above the viewport.
     */
    r -= this->row_off + this->file_off;
    if( r < 0 )         /* Desired row is above viewport. */
    {
        while( ++r <= 0 )
            if( !e_vscroll(this, -1, 1) )  /* scroll text down */
            {
                rv = 0;
                goto exit;
            }
    }
    else if( r >= nrows )
        for( r -= (nrows-1); --r >= 0 ;)
            if( !e_vscroll(this, 1,1) ) /* scroll text up */
            {
                rv = 0;
                goto exit;
            }

    /* The desired cursor is now in the viewport. Go there. */

    b_gotorc(b, dst_r - this->file_off, dst_c);
    if( this->update )
        v_gotorc(v, dst_r - this->row_off - this->file_off,
                                dst_c - this->col_off );
exit:
    monitor(this, UPDATE);
    return rv;
}

/*---------------------------------------------------------------------*/
static int bottom(win *const this, int offset )
{
    /* If offset isn't zero, set bottom to its current value + offset.
     * Otherwise set it to the current line.
     */
    if( !offset )
    {
        int lineno = b_row(&this->b) + this->file_off;
        if( lineno > this->bottom )
            this->bottom = lineno;
    }
    else if( (this->bottom += offset) < -1 )
        this->bottom = -1;

    return this->bottom;
}
/*---------------------------------------------------------------------*/
void e_replace (win *const this,int c)
{
    bottom(this,0);
    if( c )
    {
        this->dirty = 1;
        if( this->update )
            v_putc( &this->v, c, 0 );
        *b_current(&this->b) = c;
    }
    monitor(this, UPDATE);
}
/*---------------------------------------------------------------------*/
void e_ins(win *const this, int c )
{
    viewport *const v = &this->v;

    bottom(this,0);
    b_insert_c( &this->b, 1, c ? c : ' ');
    this->dirty = 1;
    if( this->update )
    {
        v_scroll_region( v, 0, -1, v_col(v),     v_row(v),
                                   v_ncols(v)-1, v_row(v) );
        v_putc( v, c ? c : ' ', 0 );
    }
    monitor(this, UPDATE);
}
/*---------------------------------------------------------------------*/
void e_del(win *const this)
{
    viewport *const v = &this->v;
    textbuf  *const b = &this->b;
    int vrow = v_row(v);
    int vcol = v_col(v);
    int brow = b_row(b);
    int bcol = b_col(b);

    b_delete_c( &this->b, 1, ' ' );
    this->dirty = 1;

    if( this->update )
    {
        v_scroll_region( v, 0,  1,      v_col(v), v_row(v),
                                  v_ncols(v)-1, v_row(v) );
        v_gotorc (v, vrow, v_ncols(v)-1);
    }
    b_gotorc (b, brow, (v_ncols(v)-1) + this->col_off);

    if( this->update )
    {
        v_putc   (v, *b_current(b), 0);
        v_gotorc (v, vrow, vcol);
    }
    b_gotorc(b, brow, bcol);
    monitor(this, UPDATE);
}
/*---------------------------------------------------------------------*/
int e_closeline(win *const this)
{
    viewport *const v = &this->v;
    textbuf  *const b = &this->b;
    int    ecol = e_col(this);  /* absolute row and column */
    int    erow = e_row(this);
    char   *new;
    int    i, bottom_line_of_buffer;
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
    if( this->bottom < 0 )
        return 0;

    b_gotorc ( b, b_row(b), 0  );
    del_line ( this, b, this->file_off + b_row(b) );
    this->dirty = 1;

    b_closedown( b, ' ');
    if( this->update )
        v_scroll_region( v, 1, 0, 0, v_row(v), v_ncols(v)-1,
                                               v_nrows(v)-1 );

    bottom(this, -1);   /* Bottom line moves up one notch
                         * because of the delete. */
    /* Import a new bottom line into the buffer if there's an import func-
     * tion and there is text in the file below the bottom line of the
     * buffer. Note: since we just deleted a buffer line, all the file-
     * relative line numbers have changed, so the line being imported is
     * actually the last line in the buffer, not the line below the last
     * line as is the case in the vertical-scroll function.
     */
    bottom_line_of_buffer = this->file_off + b_nrows(b) - 1;

    if( this->import && this->bottom > bottom_line_of_buffer )
    {
        if(new = (*this->import)(this, bottom_line_of_buffer) )
        {
            /* Move the imported line into the buffer. */

            b_gotorc(b, b_nrows(b)-1, 0 );
            for( i = b_ncols(b); --i >= 0 && *new ;)
                *b_advance(b) = *new++;
        }
    }

    /* If original row is now off the bottom of the file,
     * adjust it down to compensate.
     */

    if( erow > this->bottom )
        erow = this->bottom < 0 ? 0 : this->bottom ;

    /* Update the viewport to display the imported line. */

    if( this->update )
    {
        b_gotorc(b, v_nrows(v)-1 + this->row_off, this->col_off );
        v_gotorc(v, v_nrows(v)-1, 0 );
        for( i = v_ncols(v); --i >= 0; v_putc(v,*b_advance(b),1) )
            ;
    }

    e_gotorc(this, erow, ecol );
    e_gotorc(this, e_row(this), e_first(this) );
    monitor(this, UPDATE);
    return 1;
}
/*-------------------------------------------------------------*/
int e_openline(win *const this, int below)
{
    /* Open the line on which the cursor is resting by moving all
     * following lines down a notch and importing a blank line. If below
     * is true, open the line beneath the current one. Cursor ends up at
     * far left of new line.
     */
    textbuf  *b  = &this->b;
    viewport *v  = &this->v;
    int line     = e_row(this);
    int scrolled = 0;
    int row;

    this->dirty = 1;
    if( below )
    {
        /* If we're inserting below the last line of the file, and the
         * cursor's resting on the bottom line of both the viewport and
         * the buffer, force a scroll up without importing. In any event,
         * adjust the target line to be the one beneath the current on.
         */
        if( (line == this->bottom) && v_ateob(v) && b_ateob(b) )
            scrolled = e_vscroll( this, 1, 0 );
        ++line;
    }
    e_gotorc(this, line, 0); /* Go to line that will be replaced. */

    /* Export the bottom line of the buffer if it has something in it.
     */
    if( this->bottom >= b_nrows(b)-1 && !scrolled )
    {
        row = b_row(b);
        b_gotorc  (b, b_nrows(b)-1, 0);
        export    (this, b, this->file_off + b_row(b) );
        b_gotorc  (b, row, 0);
    }

    /* Scroll everything below current line down a notch and put cursor
     * at start of new line.
     */
    b_opendown(b, ' ');
    if( this->update )
        v_scroll_region(v, -1, 0, 0, v_row(v), v_ncols(v)-1,
                                               v_nrows(v)-1 );
    e_gotorc(this, line, 0);

    /* Update file size and tell file manager to insert line at current
     * position. */

    bottom(this, +1);   /* Update the file size. */
    return (this->insert) ? (*this->insert)( this, line ) : 1 ;
}
/*-------------------------------------------------------------*/
void e_cleartoeol( win *const this )
{
    this->dirty = 1;
    b_cleartoeol( &this->b, ' ');
    if( this->update )
        v_cleartoeol( &this->v );
    monitor(this, UPDATE);
}
```

Circle 26 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Libraries revisited

ALIBerating assembly language library with tips for management, storage, and maintenance

By John Otken

When I last wrote about assembly language libraries two years ago (*PJ* 7.5), I had an extensive library of more than 200 routines. I didn't want to release that version at the time, because I planned major improvements that would make it much easier to use.

The biggest change I made was to pattern the library after well-known models such as the standard C library, the X Window System, and Microsoft Windows. Each subsequent article featured rewritten routines from my old library according to the design of these systems. To make the distribution disk for an article, I would cut and paste low-level routines from previous articles into several miscellaneous library modules. This saves disk space but wastes time; from now on, I will distribute the entire library with each article.

A good library is more important to assembly language programmers than to C programmers, because assembly language works at a lower level. An assembly language program uses many more subroutine calls than its C equivalent, and many subroutines are shared by several programs. For this article, I have collected my library routines from previous issues of *PJ*. Including a few additions and modifications, I have about 66 files with more than 5,000 lines of code and about 120 usable functions. For lack of a better name, I refer to my assembly language library as ALIB.

For the following discussion, the first group of functions is organized according to the header files in the standard C library. These function names are listed but not described because of their similarity to the C library's functions.

John Otken is president of Soft Advances, where he hacks assembly language into debuggers and develops other software for embedded systems. He can be contacted by writing to him at P.O. Box 184, Orinda, CA 94563.

STANDARD C FUNCTIONS

The library contains all the character-testing functions from CTYPE.H, including **isalnum, isalpha, iscntrl, isdigit, isgraph, islower, isprint, ispunct, isspace, isupper, isxdigit, tolower,** and **toupper.** These functions accept a character in AL and return the results of the test in the zero flag:

```
;;      strskp space
;
;       entry  DS:SI  string pointer
;       exit   SI     updated past white space
;       uses   AX
;
strskp_space proc
ssp1:   lodsb
        call   isspace
        je     ssp1
        dec    si
        ret
strskp_space endp
```

Functions and macros from STDIO.H include FILENAME_MAX, **fclose, fopen, fread, fwrite, perror, putchar, puts, remove, rename, rewind, stderr, stdin,** and **stdout.** The library also includes several functions that are closely related to STDIO.H functions: **putchar_newline, putchar_space, puts_dgroup,** and **put_string. puts_dgroup** writes a DGROUP string without loading DGROUP into DS. **put_string** is like **puts,** but it does not append a newline:

```
mov     bx,stdout           ; send "hello world"
mov     cx,size hello_world ; to console
lea     si,hello_world
call    fwrite
```

Functions and macros from STDLIB.H include EXIT_FAILURE, EXIT_SUCCESS, **calloc, exit, free, malloc, rand,** and **realloc.** Two related functions are **exit_with_failure** and **exit_with_success:**

```
mov   cx,100        ; allocate 100 bytes
call  malloc
jc    fool          ; if not enough memory
```

The four string-handling functions from STRING.H are **strcmpi, strcpy, strlen,** and **strncpy.** Other string functions include **strcpy_limit, strlenz, strskp, strskp_line,** and **strskp_white. strlenz** is similar to **strlen,** but counts the delimiting \0 in the string length. **strskp** skips to the end of string, **strskp_line** skips to the next line, and **strskp_white** skips over spaces and tabs.

ALIB error routines are different from those in the C library. A C program or library function sets an error code in

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

the global variable **errno.** To access the error message associated with the code, the program calls **strerror.** Assembly language programs need more flexibility, so ALIB uses an error address instead of an error code. The address points to an error string. A program can access the error address via three procedures: **set_strerror** sets an error string address; **get_strerror** returns an error string address; and **clear_strerror** zeros the current error string address.

WINDOWS FUNCTIONS

The second group of functions is based on the Microsoft Windows API. To avoid conflict with the API's symbol names, I altered the function names. They are all lower case, and an underscore separates words: **FindAtom** becomes **find_atom.** (I cannot imagine using these routines in a real Windows program, so I'm not sure the name mangling is really necessary.)

The windows hashing routines include **add_atom, add_binary_atom, find_atom, find_binary_atom, get_atom_name,** and **init_atom_table. add_atom** returns a different number for each unique string. **find_atom** searches for a string and returns its number. **get_atom_name** returns the string associated with a number.

Windows memory-management functions include **global_alloc, global_calloc, global_free, global_lock, global_realloc,** and **global_unlock.** The **alloc, calloc, free,** and **realloc** functions are similar to their counterparts in the C library, with the important difference that the Windows routines work with storage handles, and the C library functions work with storage pointers. To dereference a handle, call **global_lock.** After using the pointer, call **global_unlock.** When a handle is unlocked, the system is free to move or swap the storage, allowing Windows to simulate virtual memory even on an 8088.

A Windows program calls **fatal_app_exit** to abort a program. Use this function for very serious errors only, not for simple syntax or command errors. **Get_input_state** tests for mouse or keyboard messages.

A GRAB BAG OF FUNCTIONS

These last functions do not follow any particular model, but they are useful in many assembly language programs.

You can replace a long list of PUSH and POP instructions with a single call to **save_most.** This library routine pushes all the registers on the stack except AX and automatically restores them when the procedure returns. To accomplish this feat, **save_most** does some tricky manipulations of the system stack:

```
call save_most ; this is functionally equivalent
...            ;  to the code below
ret

pushm bx,cx,dx,di,si,ds,es
...
popm  es,ds,si,di,dx,cx,bx
ret
```

A simple approach to file I/O is to read the entire source file with a single DOS call. This simplifies the program, because the code does not have to manage file buffers or boundaries. The only check is for end of file. The **read_entire_file** routine in ALIB handles this file operation.

The block storage routines for file I/O organize RAM, EMS, and disk into 16K objects. *File objects* move between disk and RAM, and *memory objects* move among RAM, EMS, and a swap file. All resources are managed on a least-recently-used basis. The block routines permit a program to manage multiple megabytes of file and memory information efficiently. For more information, see "Real-mode virtual memory," *PJ* 9.1.

The ALIB library includes 10 functions for network communications over NetBIOS: **netbios_add_name, netbios_call, netbios_cancel, netbios_check, netbios_delete_name, netbios_hang_up, netbios_receive_wait, netbios_send_broadcast, netbios_send_datagram,** and **netbios_send_wait.** The network article ran only two issues ago (*PJ* 9.4), so you may have it lying around.

A few of the common file primitives are **close_file, close_file_cf, current_file_size, open_input_file, open_output_file, move_file_pointer, read_from_file,** and **write_to_file.** The only tricky one of the bunch is **close_file_cf,**

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

which is used to close a file after an error. It returns Cf = 1 if Cf was sent on entry (because of a previous error) or if the file close operation fails.

The startup code for the ALIB library has evolved over the past two years into STARTUP2.ASM. This code works like the Microsoft C startup code: it initializes the registers, parses command-line arguments, zeros BSS, executes initialization code, and calls **main**. When **main** returns, the startup code executes terminators and then returns to DOS.

Initializers and terminators are a clever way for a program to execute code before and after **main** executes. Because the linker builds the tables, code can be executed only when a program contains a certain module. For more information, see *PJ* 9.1.

That's all for ALIB. Now let's take a look at some utilities that ease the task of managing libraries.

Turn your C/C++ compiler into a powerful scientific programming tool...
C/Math Toolchest
Now you can develop sophisticated scientific and engineering applications using your favorite C or C++ compiler. With a broad range of mathematical functions, the C/Math Toolchest™ gives C as much number crunching power as FORTRAN. More than 135 functions provide support for:
Probability and Statistics: including uniform, normal, binomial, Poisson, and hypergeometric distributions, and least-squares regression.
Complex Arithmetic and Linear Algebra: including a comprehensive set of vector and matrix operations, and solutions to simultaneous linear equations.
Numerical Analysis and Digital Signal Processing: including integration, differentiation, interpolation, root finding, convolution, FFTs, power spectrum analysis, and filter design.
To graphically view the results of your number crunching, we've also included GRAFIX™, a graphical data analysis program. The GRAFIX™ program makes it easy to edit, plot, interpolate, or perform regression analysis of your data.
You may also purchase the C source code for the library and GRAFIX™ program. The C/Math Library Source works with any ANSI standard C or C++ compiler. The C/Math GRAFIX Source works with the C and C++ compilers from Mix®, Borland®, and Microsoft®. Prebuilt libraries for the DOS versions of these compilers are included with the C/Math Toolchest.
Now Only $29.95!
Includes 430 page manual
Order Coupon
Name
Street
City
State Zip
Telephone
Send me free brochures for all Mix products
Paying By: Money Order Check
Visa MC AmX Disc.
Card#
Exp. Date
Disk Size: 5 1/4" 3 1/2"
Qty. Product Price Subtotal
C/Math Toolchest $29.95
C/Math Library Source $10.00
C/Math GRAFIX Source $10.00
Add Shipping ($5 USA, $10 Canada, $20 Foreign)
Texas Residents Add 8.25% Sales Tax
Total Amount of Your Order
Order now by calling our toll free number or mail the coupon to:
Mix Software
1132 Commerce Drive
Richardson, TX 75081
1-800-333-0330
60 Day Money Back Guarantee
Not Copy Protected ■ No Royalties
For technical support, please call:
1-214-783-6001
FAX: 1-214-783-1404
MIX software
C/Math Toolchest and GRAFIX are trademarks of Mix Software, Inc. Mix, Borland, and Microsoft are trademarks of the respective companies.
Circle 27 on Reader Service Card

WHAT'S IN A NAME?

In my first article (*PJ* 8.4), I discussed conventions for local labels within a procedure. I advise against the common practice of using long text labels within a procedure. Comments are a much better way to document conditional jumps, and, unlike long labels, comments don't waste a line. Spend more time on useful comments and procedure names and use simple local labels:

```
loop_here_more_string:  ; this technique
    lodsb               ;  is not
    cmp   al,0          ;  recommended
    jne   loop_here_more_string
```

A good syntax for local labels is three characters followed by a number (e.g., **foo1, abc23**). If the procedure name has three words, form labels from the three initials. Otherwise, try to derive the best three-letter abbreviation for the procedure name.

You might consider any special syntax that your assembler supports for local labels. For example, OPTASM supports local labels of the form **1$, 24$,** or **265$.** Another advantage of special syntax labels (and, to a lesser degree, three-initial labels) is that they are easy to identify in a program. You will never confuse a local label for a procedure name.

"Sequenced" local labels can get out of sequence, because as you work on a procedure, you inevitably add and delete labels. Assembly language programmers are normally (abnormally?) perfectionists, so this messiness will not do. Enter the ORDER program.

COME TO ORDER

The ORDER utility sequences the local labels in an assembly language program. It scans each procedure's local labels, sequences them, and writes the updated source code back to disk.

The ORDER program (originally in *PJ* 8.2) is an excellent example of multisegment assembly language programming. Unlike the average C code, well-designed assembly language programs do not need many segment overrides to access large amounts of data.

A few restrictions keep ORDER simple. Local labels must be in one of two forms: three characters followed by a number (**abc12, rnf8**) or OPTASM-style "buck" labels (**29$, 0$**). For high performance and programming

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ease, ORDER reads the entire source file in a single operation. This limits the file size to 64K.

A single DOS command can sequence all the assembly language source files in a directory when you use ORDER. After loading the library modules into the \ALIB directory, enter the following line:

```
for %c in (*.asm) do order %c
```

If you take the time to create high-quality header comments, ASMHDR's document file is all you need for using the library.

CHECK THE FILES

Maintaining control files for MAKE and LIB is a hassle. A better way would be to store all library source files in a single directory and let a utility keep the control files up-to-date. LIBGEN is such a utility.

LIBGEN sorts a list of source files in a directory and writes control files for MAKE, LIB, and OPTASM. To update a library, execute these commands with a batch file:

```
libgen *.asm
make lib.mak
lib @lib.arf
```

This works, but it leaves lots of OBJ files cluttering the directory. A better, faster way to maintain a library requires two tools from SLR Systems: OPTASM and OPTLIB.

OPTASM is still the best assembler for non-386 work. It has a mode in which it reads a **make**-like command file and assembles only out-of-date files. Because the assembler runs only once, it can cache **include** files and save lots of disk I/O. OPTLIB is an enhanced library manager. It works much better than Microsoft LIB and includes a time-based **replace** command that is perfect for libraries. Use the following batch commands with OPTASM and OPTLIB:

```
libgen *.asm
optasm @lib.opt
optlib alib -+*;
erase *.obj
```

If you must use MASM or TASM, you can still avoid OBJ clutter with a good MAKE program. I recommend Opus Make, which knows how to read the time stamps in a LIB file. You will still want a copy of OPTLIB. For this approach, LIBGEN will require a few modifications.

HELP WITH DOCUMENTATION

Another great utility for library maintenance is ASMHDR (see *PJ* 7.5). It reads assembly language source files, extracts the header comments from certain procedures, sorts them, and writes them to a document file. If you take the time to create high-quality header comments, ASMHDR's document file is all you need for using the library.

An enhanced version of ASMHDR is now available. It looks for public statements at the beginning of a source file and includes headers only for public procedures. To include all headers, use the **-a** option.

ASMHDR uses a simple parser for PUBLIC statements. The program skips initial lines that begin with tab characters (**include** statements), skips blank lines, and then looks for public lines. A public line must begin with **"\tpublic\t"**. When ASMHDR finds the first nonpublic line, it scans the rest of the file for header comments. Any public statements beyond the top of the file are ignored.

Source code for all library routines and for ASMHDR, LIBGEN, and ORDER are on the code disk for this issue. Information about Opus Make, OPTASM, and OPTLIB is provided at left. PJ

VENDOR INFORMATION

OPTASM and OPTLIB
SLR Systems
1622 North Main Street
Butler, PA 16001
(412) 282-0864
Circle **212** on Reader Service Card

Opus Make
OPUS Software
1032 Irving Street, Ste. 439
San Francisco, CA 94122
800-248-6787, (415) 664-7901
Circle **213** on Reader Service Card

New Version 4.0

Commenting Disassembler!

SOURCER™ 486

- **See how programs work**
- **Easily modify programs**

"Sourcer is the best disassembler we've ever seen." PC Magazine

SOURCER™ creates commented source code and listings from executable files or memory, suitable for reassembly. Sourcer helps clarify the inner workings of programs, with detailed in-line comments on interrupts, subfunctions, I/O ports functions, and much more. It offers complete support for all 8088/87 to 80486 instructions and V20/V30.

Sourcer provides the most comprehensive automatic analysis available to accurately separate code and data. It determines data types, uses descriptive labels for BIOS and PSP data, and links data items across multiple segments. Now supports MASM 6.0, TASM and DOS 5.0.

Professionals consistently chose Sourcer because of its ability to achieve far superior results with the least effort.

BIOS SOURCE

for PS/2, AT, XT, PC and Clones

- **Change and add features**
- **Clarify Interfaces**

The BIOS Pre-Processor ™ augments Sourcer to obtain commented listings for any BIOS ROM in your PC. Now you can understand how your specific BIOS works. Adds over 75K of comments specific to your BIOS. Tracks and identifies multiple interrupt branches with special labeling such as "int_10_video." Full automatic operation.

ASM ProPak™

Combine our advanced assembly language tools and save up to $120! All ASM ProPaks include Sourcer & the BIOS Pre-Processor, Unpacker™ for unpacking packed EXE files, and View-It ™ the highest speed file viewer providing a 132 colum display. ProPak II adds ASMtool ™ 486 to flow chart 8088 to 80486 source code. ProPak III adds ASM Checker the only assembly source code bug finder.

Sourcer 486 - Commenting Disassembler	$129.95
BIOS Pre-Processor	49.95
Sourcer w/BIOS - (save $10)	169.95
Unpacker - Unpacks packed files	39.95
View-It - View 132 column files	69.95
ASM ProPak I - All above (save $40)	249.80
ASMtool 486 - Automatic flowcharter	199.95
ASM ProPak II - All above (save $90)	399.75
ASM Checker - Finds source code bugs	179.95
ASM ProPak III - All above (save $120)	549.70
Memory Commander - Get up to 920K of low DOS memory and much more	99.95

Shipping: USA $6, Canada/Mexico $10; Other $18. CA residents add sales tax. © 1991 VISA/MasterCard/COD accepted.

30-DAY MONEY-BACK GUARANTEE

1-800-648-8266

V Communications, Inc.
4320 Stevens Creek Blvd., Suite 275-PJ2
San Jose, CA 95129 FAX 408-296-4441
408-296-4224

Contact this advertiser directly

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Save your video mode

Preserve an EGA or VGA display mode by wrapping VIDSAVE and SETMODE into DOS

By Walter Oney

Some of my favorite programs have the nasty side effect of trashing my VGA video mode when they exit. If I start out with, say, a 50-line display of white characters on a blue background and run a full-screen application that was built before EGA and VGA displays became popular, the application isn't apt to restore the display attributes when it exits. Instead, it leaves me with a black window in the top half of the screen or with a black screen displaying only 25 lines of text.

I have written a short program, VIDSAVE.COM, that you can use with many DOS applications to save and restore the text mode attributes of your EGA or VGA display. The VIDSAVE code can also be incorporated into your text application to preserve VGA text attributes. Suppose, for example, you've installed XTreePro (XTG.EXE), from Executive Systems. To encapsulate VIDSAVE with XTG, copy VIDSAVE.COM into the same directory under the name XTG.COM.

When you try to start XTG, DOS finds the COM file instead of the EXE file. This causes DOS to run VIDSAVE whenever you run XTreePro. VIDSAVE captures the current video state, runs XTG.EXE, and restores the original video state before returning to DOS.

Walter Oney is a principal software engineer at Rational Systems Inc., 220 North Main Street, Natick, MA 01760, where he is responsible for Windows-related products.

FIRST THINGS FIRST

VIDSAVE.ASM (Listing 1) is written as a COM file, because DOS searches for a COM before it searches for an EXE file. As is the case with other COM programs, when VIDSAVE gains control the segment registers all point to a single 64K segment. Because it remains in memory while the main application runs, the first thing VIDSAVE does is use DOS function 4Ah to shrink its segment down to the smallest possible size.

One of the consequences of writing VIDSAVE as a COM file is that it can't contain any segment fixups. However, the parameter list for DOS function 4B (EXEC) needs to contain three far pointers, so VIDSAVE must relocate them explicitly.

After taking care of these housekeeping details, VIDSAVE determines the full pathname of the file that DOS actually loaded. In the XTreePro example, the name is something like C:\TOOLS\XTG.COM. To derive the name of the target program you've encapsulated, VIDSAVE replaces the COM extension with EXE.

Before DOS 3.3, a program had no way to determine what its name was. Beginning with 3.3, the full pathname of the executing program is appended to the environment strings pointed to by the program segment prefix (PSP). Each environment string ends in a null byte; the final string ends in an extra null byte. After that point in memory, there is a two-byte count field followed by the full program pathname, which also ends in a null byte.

PARAMETER UPKEEP

The heart of VIDSAVE is the local subroutine **vidrest,** which appears near the end of the program. **vidrest** restores the original video mode on request. The half-dozen parameters needed to describe the original mode are captured in the code after the **getmode** label. It's easiest to explain these parameters in terms of the local variables from which **vidrest** retrieves their values.

vidmode holds the value of the BIOS video mode. This value is normally 3, which denotes a color text mode having 25 lines of 80 characters. With EGA and VGA adapters, mode 3 also includes 43- or 50-line formats, respectively. INT 10h, function 0Fh returns this value in the AL register. **vidrest** restores it by using INT 10h, function 00h.

vidpage holds the index of the active display page, which is usually zero. This value is supplied by some of the parameter-setting calls that **vidrest** uses. The same INT 10/0F call that determines the video mode also returns this index in the BH register.

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

vidcurs contains the start and end scan lines for the character cursor. INT 10h, function 03h returns this in the CH:CL register pair. Later on, **vidrest** will load these values into CH:CL so that it can call INT 10h, function 01h to restore the original cursor shape.

vidfont holds one of the values 1111h, 1112h, or 1114h. These are AX values for INT 10h, and they designate the 8×14 font (28 lines on a VGA, 25 on an EGA), the 8×8 font (50 lines on a VGA, 43 on an EGA), or the 8×16 font (25 lines on a VGA), respectively. To determine which of these three choices is appropriate for the current font, VIDSAVE uses INT 10h, function 1130h. This function returns the height of the current font in the CX register. (Actually, it returns the number of bytes needed for each character in the internal character generator table. Because eight bits are needed for each row of a character, this number equals the character cell height.)

charattr contains the character attribute byte (background and foreground colors) that **vidrest** uses when it clears the screen. This attribute byte is determined by using INT 10h, function 08h, which reads the attribute at the current cursor location. (If that position contains an atypical attribute, the screen restoration may produce an unexpected result.) **vidrest** uses INT 10h, function 06h to clear the entire screen. The number of the bottom row is found in the BIOS data area at absolute location 40:84, and the number of screen columns is found at location 40:4A.

border is the color of the screen border. An EGA display has no way to determine this color, so VIDSAVE uses the background color for the current character. INT 10h, function 1Ah can be used to determine whether the adapter is VGA. If it is, INT 10h, function 1008h retrieves the border color. **vidrest** uses INT 10h, function 1001h to set the border color back to its original value.

RIGHT ON TARGET

VIDSAVE uses the DOS function 4Bh (EXEC) to execute the target appliction. While the target application is executing, VIDSAVE monitors INT 21h for function 4Bh (EXEC) calls and INT 10h for function 01h (**Set Cursor Type**).

Hooking INT 21h and looking for 4Bh functions allows VIDSAVE to restore the original video mode when you shell out to DOS from the encapsulated application. I appreciate this feature when using my venerable old text editor, which would otherwise clear the screen to 25 black lines. If your application does EXEC calls to run commands that don't produce any output or if it doesn't restore its own screen when it regains control, you may want to disable this function. If you don't, you might see undesirable screen flashes—or worse things.

The technique of wrapping an EXE program with a COM file is potentially very powerful.

Hooking INT 10h allows me to compensate for another defect of my trusty text editor (which I can't seem to give up). The editor sets the character cursor "type" to the apparently nonsense values of 0600h and 0200h. Literally, these mean "start on row 6 [or 2] and end on row 0," but many video BIOS ROMs substitute the largest possible end row to make sense of the request. On one of the machines I use, however, the BIOS hides the cursor instead. To work around the problem, I added the INT 10h hook and code to substitute meaningful values for the cursor's start and end rows.

INITIAL MODE

Before you go to the bother of downloading and building VIDSAVE, you'll want to establish your own nonstandard ambient video mode. There are several ways to do this. You can use a package such as the Norton Utilities, or you can rely on escape sequences for ANSI.SYS. I use a short program I call SETMODE.COM that I created with the DEBUG script shown in Figure 1.

SETMODE puts a VGA display into 50-line mode (or an EGA into 43-line mode), makes the screen border blue, and establishes a blue screen in which default character output will be white. It also sets the character cursor to a height of three pixels, which improves its appearance on some monitors.

SUIT YOURSELF

The technique of wrapping an EXE program with a COM file is potentially very powerful. As my experience with the cursor-type BIOS problem shows, it enables you to write a wrapper that will intercept almost any interrupt and correct the parameters it uses before passing it along to DOS or the BIOS.

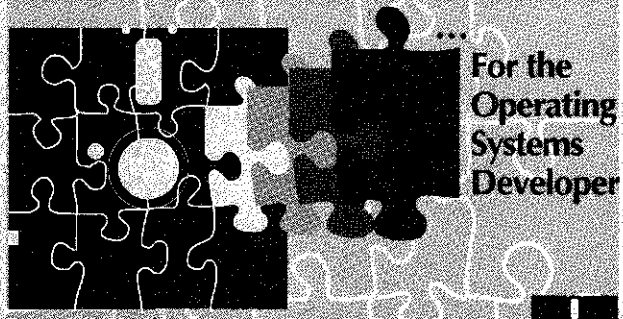
I tailored SETMODE to be specific to my own hardware and taste. A product-grade mode-setting program would inquire whether an EGA or VGA adapter is present, and it might also let the user specify colors and font size by command-line arguments or by some other method. A good book on video programming will help you adapt SETMODE to your own needs, however, and you may find (as I did) that generality is less important than being able to write a simple program quickly that you can use on one machine forever. PJ

FIGURE 1: DEBUG script

```
C>debug setmode.com
File not found
-a 100
2974:0100       MOV     AX,0003
2974:0103       INT     10
2974:0105       MOV     AX,1112
2974:0108       XOR     BL,BL
2974:010A       INT     10
2974:010C       MOV     AH,01
2974:010E       MOV     CX,0507
2974:0111       INT     10
2974:0113       MOV     AX,1001
2974:0116       MOV     BX,0100
2974:0119       INT     10
2974:011B       MOV     AX,0040
2974:011F       MOV     ES,AX
2974:0120       ES:
2974:0121       MOV     DX,[004A]
2974:0125       DEC     DX
2974:0126       ES:
2974:0127       MOV     CH,[0084]
2974:012B       1C9     XOR
2974:012D       MOV     BX,1F00
2974:0130       MOV     AX,0600
2974:0133       INT     10
2974:0135       MOV     AX,4C00
2974:0138       INT     21
-r cx
CX 0000
:3a
-w
Writing 003A bytes
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PUTTING IT ALL TOGETHER...



...For the Operating Systems Developer

UNLEASH THE POWER OF A REAL-TIME MULTITASKING DOS IN YOUR EMBEDDED SYSTEM!
Embedded DOS-- The full-featured, MS-DOS 3.31-compatible DOS that packs power and performance matching real-time executives. Royalties $6/copy.
GS #401, Embedded DOS Adaptation Kit..... $495/kit

WRITING DEVICE DRIVERS IS A SNAP WITH DEVICE DRIVER TOOLKIT!
With the complete source to Embedded DOS's device drivers, you'll have a head start on your next device driver project with this model code!
GS #407, Embedded DOS Device Driver Toolkit... $175/kit

BUILD UP YOUR OWN ARSENAL OF UTILITY PROGRAMS WITH OUR UTILITY TOOLKIT!
Explore and enhance these Embedded DOS utilities to build your own custom environment! Full source code in C included.
GS #408, Embedded DOS Utility Toolkit..... $195/kit

BOOT YOUR OWN APPLICATIONS WITH BOOTER TOOLKIT!
This fascinating software construction set provides the tools you need to develop bootable systems for the PC. Includes multitasking scheduler, file system, window manager, and full source code in C and assembly language.
GS #201, Booter Toolkit..... $149/kit

NEW FROM THE NATIONAL SECURITY AGENCY (NSA)-- COMPUTER SECURITY STANDARDS AND GUIDELINES!
Through an arrangement with NSA, we are pleased to offer a fascinating list of 19 volumes covering operating system security for handling classified and sensitive data. Call for complete title list. (Shipped to U.S. customers only.)
GS #701, DoD Trusted Computer System Evaluation Criteria.....$20.
GS #702, DoD Password Management Guidelines.....$20.
GS #703, DoD Computer Security Requirements (2 volume set)....$40.
GS #704, ACL Features for Trusted UNIX.....$20.
GS #705, Trusted Network Interpretation (two volume set).....$40.
GS #799, Security Handbook Bundle (incl. all 19 volumes).....$275.

GENERAL SOFTWARE
P.O. Box 2571
Redmond, WA 98073
(206) 391-4285 Phone
(206) 746-4655 Fax

To receive a free product catalog or to place an order, please call (206) 391-4285. VISA/MasterCard/COD orders gladly accepted.

Embedded DOS, Booter Toolkit, General Software and the GS logo are trademarks of General Software, Inc. MS is a registered mark of Microsoft. Unix is a trademark of AT&T. Copyright (C) 1991 General Software, Inc. All rights reserved.

LISTING 1

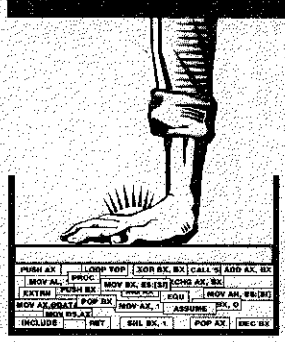
```
;    VIDSAVE.COM -- Wrapper for ill-behaved programs that wipe out
;                   your 50- or 43-line video display mode
;
;    To use this program, copy it to the directory where the .EXE for
;    the target application lives. Rename this .COM file to have the
;    same file name (but still a .COM extension).
;
;    Copyright (c) 1991 by Walter Oney, Rational Systems, Inc.

pushall  macro
         push  ax
         push  bx
         push  cx
         push  dx
         push  si
         push  di
         push  bp
         endm
popall   macro
         pop   bp
         pop   di
         pop   si
         pop   dx
         pop   cx
         pop   bx
         pop   ax
         endm

         name  vidsave

vidsave  segment byte public 'code'
         assume cs:vidsave, ds:vidsave
         org   100h                ; required for .COM file usage
;    Initialization. Establish reasonably sized stack and shrink
;    memory block down.

begin:   lea   sp, topsp           ; Set new stack top.
         lea   bx, topsp+15        ; Compute # para's in shrunk block.
         mov   cl,4
         shr   bx, cl              ;   ..
         mov   ah, 4Ah             ; Resize memory block.
         int   21h                 ;   ..

         mov   seg1, ds            ; Set segment address where needed.
         mov   seg2, ds            ;   ..
         mov   seg3, ds            ;   ..

         mov   ah,30h              ; Test for DOS 3.3 or greater.
         int   21h
        xchg   ah, al
         cmp   ax,0303h
         jae   dos_ok

         lea   dx,old_dos_msg      ; Indicate that we need DOS 3.3.
         mov   ah,9
         int   21h
         mov   ax,4c01h            ; Terminate with error.
         int   21h
;    Locate full pathname of executed command at the end of the
;    environment block. Replace the .COM extension with .EXE.
dos_ok:  cld                       ; forward direction
         mov   es, word ptr ds:[2Ch] ; Point ES:DI to start of environment.
         xor   di, di              ;   ..
         xor   al, al              ; Get zero to compare against.
         xor   cx, cx              ; Set practically infinite count.
         dec   cx                  ;   ..

skipvar: repne scasb               ; Find end of environment variable.
         scasb                     ; Compare next byte.
         jne   skipvar             ; If not another null, do it again.

         add   di, 2               ; Skip word count.
         mov   execpath, di        ; Save ptr to start of program name.
         repne scasb               ; Find end of name.
         mov   byte ptr es:[di-4], 'E' ; Change extension to "EXE."
         mov   byte ptr es:[di-3], 'X' ;   ..
         mov   byte ptr es:[di-2], 'E' ;   ..

         mov   execpath+2, es      ; Complete ptr to program name.
         push  ds                  ; Set ES back to our code/data segment.
         pop   es                  ;   ..
;    Determine the current video characteristics (mode, page, font size).
getmode: mov   ah, 0Fh             ; Get video mode.
         int   10h                 ;   ..
         mov   vidmode, al         ; Save original mode.
         mov   vidpage, bh         ; Save original display page.
         mov   ah, 03h             ; Get cursor position and shape.
         int   10h                 ;   ..
         mov   vidcurs, cx         ; Save shape for later restore.
         mov   ax, 1130h           ; Get font information
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        xor   bh, bh            ;  for current font.
        int   10h               ;   ..
        mov   ax, 1112h         ; Assume we're using small font.
        cmp   cx, 8             ; Using 8 * 8 font?
        je    setfont           ; If yes, good.
        mov   ax, 1114h         ; No. Try for the 8 * 16 font.
        cmp   cx, 16            ;   ..
        je    setfont           ;   ..
        mov   ax, 1111h         ; No. Must be the 8 * 14 font.
setfont: mov  vidfont, ax       ; Save function code to restore this font.

;    If we're running on a VGA, we can determine the border color by reading
;    the palette register. With any sort of display, we can get the
;    attribute byte for any random location on the screen.
        mov   ah, 08h           ; fcn 8: read char & attr at cursor
        mov   bh, vidpage       ;   (for the current page).
        int   10h               ;   ..
        mov   charattr, ah      ; Save attribute byte for later restore.
        mov   cl,4
        shr   ah, cl            ; Assume border color is same as
        mov   border, ah        ;   background color of attribute.

        mov   ax, 1A00h         ; Issue fcn 1A (read configuration).
        int   10h               ;   ..
        cmp   al, 1Ah           ; If AL comes back != 1A, not a VGA.
        jne   notvga            ;   ..
        cmp   bl, 8             ; Is primary display a VGA with color?
        jne   notvga            ; If not, don't try to read border color.

        mov   ax, 1008h         ; Read border color.
        int   10h               ;   ..
        mov   border, bh        ; Save border color for later restore.
notvga:
;    Hook INT 10 so we can prevent the cursor from getting screwed up.
;    Hook INT 21 so we can monitor for EXEC calls and run them with
;     the original video mode.
        mov   ax, 3510h         ; Get current INT 10 & 21 vectors.
        int   21h               ;   ..
        mov   word ptr org10, bx ;   ..
        mov   word ptr org10+2, es ;   ..
        mov   ax, 3521h         ;   ..
        int   21h               ;   ..
        mov   word ptr org21, bx ;   ..
        mov   word ptr org21+2, es ;   ..
        push  ds                ; Restore ES.
        pop   es                ;   ..

        lea   dx, int10         ; Hook INT 10.
        mov   ax, 2510h         ;   ..
        int   21h               ;   ..

        lea   dx, int21         ; Hook INT 21 also.
        mov   ax, 2521h         ;   ..
        int   21h               ;   ..

; Execute the application whose file name is the same as our own file name.
        lea   bx, execparm      ; es:bs -> parameter block
        lds   dx, dword ptr execpath ; ds:dx -> pathname
        assume ds:nothing
        mov   ax, 4B00h         ; Load & execute program.
        pushf                   ; Don't go through our EXEC trap!
        call  dword ptr cs:[org21];   ..

        mov   ax, cs            ; Reset segment registers.
        mov   ds, ax            ;   ..
        mov   es, ax            ;   ..
        assume ds:vidsave       ;   ..

;    Restore the environment and terminate.
        push  ds                ; Save DS across unhook.
        lds   dx, org10         ; Unhook INT 10.
        mov   ax, 2510h         ;   ..
        pushf                   ;   ..
        call  dword ptr cs:[org21] ;   ..
```


QUANTASM • POWER • LIB:
FAST AND COMPACT, 100% ASSEMBLY LANGUAGE

NEW VERSION 2.0



"Very, very tight code."
Zoran Milenovic, Chicago

"I was productive immediately. Now I can't do without it"
Stuart McCarley, San Jose

"I can use as little or as much as I want for each project."
Rob Larson, Santa Clara

A few years ago my friend Rob said, "There should be an inexpensive, robust library for assembly language programmers. But it must be so easy to use that I can become productive immediately. I don't have the time or desire to learn someone else's style and conventions on how I ought to be programming."

That's why I began documenting my personal library as Quantasm Power Lib 1.0(QPL). Since then our customers began designing the next version so that it would be more powerful and easier to use than any other library.

Now, QPL 2.0 has more than 600 routines(doubled from QPL 1.0). QPL is very granular; you link in only what you use. We even have two ready-to-use windowing systems. One full featured system(mouse support, drop shadows, several border types, etc.) and a new compact system. (full system: 4-6K, compact system: 1.5-2K)

Some features
The window systems includes popup windows, scrolling data entry screens, data entry forms, a dozen pre-defined and customizable form fields, pick lists, vertical, horizontal and pulldown menus, overlapping and moveable windows. Command line and file name parsing. Ctrl-C and critical error handlers. String and memory block handling, numeric conversions, 32-bit math, date/time functions & math, bit manipulations, execute DOS programs, sound, data compression and more.

QPL supports any text mode screen size, a mouse, DOS extenders, memory managers, TSR's and high level languages.

- free tech support
- no cryptic macros
- no royalties
- 30-day money back guarantee

QUANTASM POWER LIB $99.95
with source code $299.95

other products:
ASMFLOW Professional $199.95
Magic TSR Toolkit $199.95
Floating Point Library $99.95

QUANTASM CORPORATION

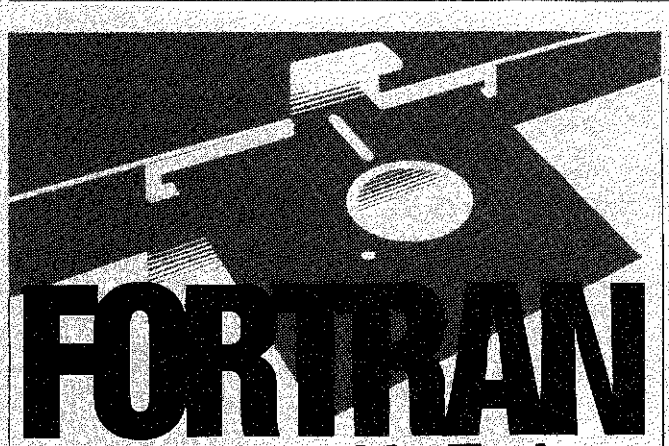
19855 Stevens Creek Blvd,
Suite 154
Cupertino, CA 95014

To order or get more info call
408 244-6826(Voice & Fax)
800-765-8086

Circle 29 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        pop   ds                ;   ..
        push  ds                ; Unhook INT 21 too.
        lds   dx, org21         ;   ..
        mov   ax, 2521h         ;   ..
        pushf                   ;   ..
        call  dword ptr cs:[org21] ;   ..
        pop   ds                ;   ..

        call  near ptr vidrest  ; Restore video environment.
        mov   ax, 4C00h         ; Terminate process.
        int   21h               ;   ..

;   INT 10 handler (needed to prevent cursor from disappearing on
;    my home machine due to probable video BIOS bug):
        assume nothing, cs:vidsave
int10:  cmp   ah, 01h           ; Look for Set Cursor Type call.
        je    setcur            ; Stay here if so.

chain10: jmp  dword ptr cs:[org10]; Chain to original 6D handler.

setcur: cmp   cx, 0600h         ; Check for start 6, end 0.
        je    fixcur            ;   ..
        cmp   cx, 0200h         ; Also for start 2, end 0.
        jne   chain10           ; Ignore all other cursor sets.
        mov   cx, 0207h         ; Set end row 7.
        jmp   chain10           ;   ..

fixcur: mov   cx, 0507h         ; Set end row 7.
        jmp   chain10           ; Chain to original handler.

;   INT 21 handler (needed to restore video during EXEC from an app):
int21:  cmp   ah, 4Bh           ;  EXEC function?
        je    exec              ;  If yes, stay here to deal with it.
        jmp   dword ptr cs:[org21]; No. Chain to original handler.

exec:   pushall                 ; Save everything.
        push  ds                ;   ..
        push  es                ;   ..
        lds   dx, cs:org21      ; Unhook ourselves during EXEC.
        mov   ax, 2521h         ;   ..
        pushf                   ;   ..
        call  dword ptr cs:[org21]; ..

        mov   ax, cs            ; Get DS addressability for VIDREST.
        mov   ds, ax            ;   ..
        call  near ptr vidrest  ; Restore video to original.

        pop   es                ; Restore everything.
        pop   ds                ;   ..
        popall                  ;   ..
        int   21h               ; Do EXEC from here.
        pushall                 ; Save registers again.
        push  ds                ;   ..
        push  es                ;   ..

        mov   ax, cs            ; Rehook INT 21.
        mov   ds, ax            ;   ..
        mov   dx, offset int21  ;   ..
        mov   ax, 2521h         ;   ..
        pushf                   ;   ..
        call  dword ptr cs:[org21]; ..

        pop   es                ; Restore registers.
        pop   ds                ;   ..
        popall                  ;   ..
        iret                    ; Return to application.

;   Internal procedure to restore the video environment:
        assume ds:vidsave
vidrest proc  near
        mov   al, vidmode       ; Restore original video mode.
        xor   ah, ah            ;   ..
        int   10h               ;   ..

        mov   ax, 0500h         ; Force display page 0.
        int   10h               ;   ..

        mov   ax, vidfont       ; Restore original font.
        xor   bl, bl            ; character block = 0
        int   10h               ;   ..

        mov   ah, 03h           ; Get cursor to reset MCGA.
        xor   bh, bh            ;   ..
        int   10h               ;   ..

        mov   al, vidpage       ; Restore original display page.
        mov   ah, 5             ;   ..
        int   10h               ;   ..

        mov   cx, vidcurs       ; Restore cursor shape.
        mov   ah, 01h           ;   ..
        int   10h               ;   ..

        mov   ax, 1001h         ; fcn 10, subfcn 01: set border color.
        mov   bh, border        ;   ..
        int   10h               ;   ..

        mov   ax, 0600h         ; fcn 06: scroll up, al=0: clear window.
        mov   bh, charattr      ; attribute to fill window with
        xor   cx, cx            ; ch/cl = top/left
        mov   dx, 40h           ; Address BIOS data area at 40:
        mov   es, dx            ;   ..
        mov   dx, es:4Ah        ; Set DL = right column.
        dec   dx                ;   ..
        mov   dh, es:84h        ; Set DH = bottom row.
        int   10h               ; Go clear the whole screen.
        ret                     ; Return to local caller.
vidrest endp
;   Data:
execpath dw   0                 ; offset to name of program to execute
execparm dw   0                 ; seg # of environment (also part of ptr)
         dw   80h               ; segment:offset of command tail
seg1     dw   ?                 ;   ..
         dw   5Ch               ; segment:offset of 1st FCB
seg2     dw   ?                 ;   ..
         dw   6Ch               ; segment:offset of 2d FCB
seg3     dw   ?                 ;   ..
vidfont  dw   ?                 ; INT 10 fcn to restore font
vidpage  db   ?                 ; video page number
vidmode  db   ?                 ; video mode
vidcurs  dw   ?                 ; original cursor shape
border   db   ?                 ; border color
charattr db   ?                 ; character attribute to restore

org10    dd   0                 ; original INT 10 vector
org21    dd   0                 ; original INT 21 vector
         db   128 dup (0)       ; presumably enough stack . . .
topsp    label byte
old_dos_msg db 'This program requires DOS 3.3 or greater',0dh,0ah,'$'
vidsave  ends
         end  begin
```

FORTRAN
Indispensable Tools
For Serious Programmers!

FORWARN
$399
- Does for Fortran what Lint does for C, and more!
- Finds mismatched argument lists and common blocks, misspelled variables and other errors
- Creates cross-reference lists and call-tree diagrams
- NEW! Summarizes routine and common variable usage

FORTRAN DEVELOPMENT TOOLS
$149
- Reformats and indents source code
- Restructures GOTOs to IF-THEN-ELSE blocks
- Creates listings with highlighted DOs and IFs
- And More! Seven tools in all!

For DOS. Also for DOS protected mode, OS/2 and Unix. 30 Day Money Back Guarantee!

Quibus ENTERPRISES, INC.
3340 Marble Terrace • Colorado Springs, CO 80906 • (719) 527-1384

Circle 30 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The midnight vampire

Banish the "lost-day" problem from your system without losing any sleep

By Murray L. Lesser

When the midnight vampire strikes, it sucks a day from the DOS calendar. It strikes only when your system has been running past midnight, but it doesn't strike consistently. An even more elusive problem—the two-midnight vampire—may infect some systems that are left running over two or more midnights in a row. Some believe that the midnight vampire (also known as the "DOS lost-day problem") is due to bugs in the code that handles the BIOS time-counter-overflow flag in a number of DOS versions [Reference 1].

Fortunately, this belief is wrong. The midnight vampire lives in application programs that operate by calling directly on the BIOS **read time** (INT 1Ah, AH=0) function, rather than by letting DOS handle all timing functions. The only "DOS version" element of the mystery is that the vampire is more likely to strike if you are using a DOS version earlier than 3.3.

ILLUSTRATION BY JIM CARPENTER

Using the short assembled demonstration program VAMPIRE.COM (Listing 1), you can test the effects of the midnight vampire for yourself. The tiny TSR GARLIC.COM (Listing 2) foils the vampire (you can prove it by rerunning VAMPIRE after you have installed GARLIC).

LATER THAN YOU THINK

Many programmers apparently do not understand the underlying architecture of the hardware system they are programming for. In particular, they don't understand that the BIOS **read time** function is for the use of the operating system only and is dangerous to the operating-system clock if INT 1Ah, AH=0 is called from anywhere except within that operating system. The danger occurs only when an offending program is the first to be executed near or just after midnight.

To review a little PC system architecture: a hardware interrupt in all IBM (and compatible) PCs updates the BIOS timer count—a double word starting at address 0:46C—approximately 18.2 times per second.

Murray L. Lesser is a retired IBM engineer. He has written three books about enhancing various Microsoft BASIC compilers with assembled modules. The most recent one, Advanced QuickBASIC 4.0, *was published by Bantam.*

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

AMX

REAL-TIME MULTITASKING KERNEL

8086/88, 80x86/88 80386
Z80, 64180, 8080/85 68000/10/20

- Fast, reliable operation
- Compact and ROMable
- PC peripheral support
- DOS file access
- C language support
- Preemptive scheduler
- Time slicing available
- Configuration Builder
- Complete documentation
- Intertask messages
- Message exchanges
- Dynamic operations
  - task create/delete
  - task priorities
  - memory allocation
- Event Manager
- Semaphore Manager
- List Manager
- InSight™ Debugging Tool

THE BEST

Join over 600 developers such as IBM®, Xerox, Hewlett Packard, Hayes, Hughes Aircraft and NASA.

CHOOSE AMX

The best low-cost, high-performance real-time multitasking system available today.

No Royalties
Source Code Included

Demo Disk and Manual only $75 US *Call for prices for*
AMX 86 $3000 US *other processors.*
(Shipping/handling extra)

IBM is a registered trademark of IBM Corp.
Z80 is a trademark of Zilog, Inc.
AMX, AMX 86, InSight are trademarks of KADAK Products Ltd.

KADAK Products Ltd.
206-1847 West Broadway
Vancouver, B.C., Canada
V6J 1Y5
Telephone: (604) 734-2796
Fax: (604) 734-8114

Circle 31 on Reader Service Card

When 1800B0h "ticks" (24 hours) have elapsed, BIOS thinks it is midnight: the counter is reset to zero, and a past-midnight flag at address 0:470 is set.

Note that according to the published ROM BIOS listings for IBM PCs from the original through the PC Convertible, the timer-overflow flag is updated by a **MOV mem, immed** instruction, not by an **INC mem** instruction. Timer-overflow flags in IBM systems always contain either a one or a zero.

The BIOS **read time** function reads the BIOS timer count. The function returns the 0:46C double-word value in registers CX:DX and returns the value of the overflow flag byte in AL. Finally, *it resets the overflow flag to zero.*

If DOS is the first caller of the BIOS **read time** function after midnight, it uses the presence of the overflow flag to increment its internal day counter (at a version-specific location). Using the INC instruction instead of adding the value of the flag byte to the day count is not a bug; it is a concession to reality. DOS cannot determine how BIOS set the past-midnight flag, so it cannot "know" how many midnights have passed since that byte was last read.

But if some application programmer has been so remiss as to reset the overflow flag before DOS sees it, a day is lost. (This may not be the programmer's fault; the compiler may have done the damage.)

HOLDING UP A MIRROR

VAMPIRE.ASM (Listing 1) does almost everything wrong. It begins by calling the "forbidden" BIOS interrupt function to get the starting time for a timing loop and continues to call it on every loop until 100 ticks (a little over five seconds) have occurred. If you want to see the midnight vampire in action, perform the following experiment.

Use the DOS TIME function to reset your system clock to five seconds before midnight. (If you are using DOS version 3.3 or later and haven't disconnected your DOS system clock from the real-time clock as described in Reference 2, this function will also reset your real-time clock.) Now run VAMPIRE. When it ends, check the DOS time and date. If you are using unadulterated DOS, TIME will indicate that you are now into the next day, but DATE will show that it is still today. (If you haven't disconnected, don't forget to reset both the time and date of your real-time clock.)

VAMPIRE.ASM is as pathological a program as you could imagine, but it is not unique. Ever since the ill-conceived **clock** function was added to Standard C, all Borland C compilers have implemented it in a manner somewhat analogous to VAMPIRE.ASM. Those compilers use the BIOS interrupt to read the timer count as part of the C startup routine. This makes it impossible to eliminate the vampire that lives in programs compiled with any Borland C compiler since Turbo C v 2.0 by avoiding all calls to the function. And if you do use **clock,** there is a midnight bug in Borland's version; it doesn't know midnight has passed unless **clock** has seen a nonzero flag byte. If the program starts execution before midnight and includes a legitimate DOS call on INT 1Ah, AH=0 after midnight (but before the first clock call), **clock** will return the wrong answer.

I single out Borland because I use Borland C++ v 2.0. There may be other equally ignorant compiler implementers, perhaps for other languages as well as for C. I would be leery of using any compiler-furnished time-measurement function unless I knew that the function compiled into bug-free code for execution around midnight.

But back to the midnight vampire, who may well live in one or more of your vendor-supplied application programs. Is there any way to fix your system to remove its effect?

The DOS implementers have tried to minimize the problem by making changes to DOS. Every version since DOS 3.3 calls the BIOS **read time** function repeatedly during COMMAND.COM idle cycles (while waiting for keyboard input) and updates its date counter if it finds the overflow flag. This minimizes the "window" (in the NASA sense) during which the vampire can strike but doesn't eliminate it completely. However, some shells that run under DOS suppress this automatic date update, thereby increasing the size of the window.

The other fix, offered with PC-DOS 5.0, is a substitute CLOCK$ driver (CMOSCLK.SYS) that uses the real-time clock, rather than the DOS clock, for all DOS "time stamp" functions. Unfortunately, this cure is worse than the disease. If CMOSCLK is installed, the minimum granularity of any DOS-using time-interval

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

measurement in your program will jump from about 55 milliseconds to 1 second, and many programs will no longer execute correctly.

STAKING OUT THE VAMPIRE

The way to foil the midnight vampire is to examine the flag byte after every BIOS **read time** call that didn't originate in DOS and rewrite all such nonzero flag bytes back into 0:470. This solution requires separating those INT 1Ah, AH=0 calls that come from DOS from those that do not. The job is handled by the tiny TSR (336 bytes, installed) shown in GARLIC.ASM (Listing 2).

GARLIC contains one potential booby trap. A BIOS **read time** call is identified as having come from DOS by looking at the contents of the DS register on entry to the new INT 1A handler. For PC-DOS, this value is 0070h, as shown. But there may be OEM-furnished MS-DOS systems that do not keep DOS data in that segment. You should test your system (or, more important, a client's system) for the DOS data area before installing GARLIC permanently.

After loading GARLIC from the DOS command line, locate the full address of the new interrupt handler. The easy way is with Quarterdeck's MANIFEST program. However, you can use any debugger to find the address from the double-word vector starting at 0:68.

Once you have the interrupt handler address, perform the small SYMDEB (or DEBUG) exercise simulated in Figure 1. When the breakpoint is reached, the DS register will show which DOS data segment is being used. (If it is not 0070h, you will have to edit GARLIC.ASM accordingly.) Unfortunately, if you are using a DOS-based debugger, the exercise will lock up your system. You probably will have to use the Big Red Switch to get your system back alive.

Once you know that GARLIC is compatible with your system configuration, you can load it from AUTOEXEC.BAT. The proof of the pudding is to rerun the VAMPIRE test. This time, when VAMPIRE ends, both TIME and DATE should show tomorrow's. (If you haven't disconnected, don't forget to reset both the time and date of your real-time clock.)

A slight modification to GARLIC.ASM will cut the forward tie (introduced in DOS 3.3) between the DOS clock and the real-time clock so that resetting either the DOS time or the date does not also reset both the real-time clock time and date. "Uncomment" the four lines indicated in the listing note, thus producing the source code for GARLIC PLUS. (An executable copy of GARLICP.COM is included on this issue's code disk.) Although this "enhancement" to GARLIC is logically equivalent to patching IBMBIO.COM (as described in Reference 2), its performance suffers by comparison—you are adding code, rather than deleting code, for every DOS **set time** or **set date** function executed.

DOUBLE JEOPARDY

Depending on your software system's configuration, you may be subject to the two-midnight vampire if you believe the superstition that hardware will last longer if you never turn it off, or if you run a

FIGURE 1: Symdeb exercise

```
F:\>symdeb
Microsoft (R) Symbolic Debug Utility Version 4.00
Copyright (C) Microsoft Corp 1984, 1985. All rights reserved.

Processor is [80286]     ;Symdeb doesn't know about the 80386.
-a100
18AD:0100 mov ah,2c      ;DOS "get time" function
18AD:0102 int 21
18AD:0104
-g b5a4:0108             ;Address of my new 'loaded high' INT 1Ah handler
AX=0000  BX=035A  CX=0006  DX=0000  SP=08E0  BP=08EA  SI=014F  DI=03B6
DS=0070  ES=0123  SS=0123  CS=B5A4  IP=0108   NV UP DI PL ZR NA PE NC
B5A4:0108 FB             STI
-
```


Debug C

What's *your* problem?

- *program hanging?*
- *memory overwrites?*
- *operating system overwrites?*
- *corrupted heap?*
- *memory leaks?*
- *dangling pointers?*
- *open file pointers?*
- *invalid calls to free()?*

If your code has a bug that's making *you* sick, *call the doctor*—**Dr. MD, the run time memory debugger for C!** Sure, you *could* find it yourself, but how much would it cost you in time, money and aggravation?

Dr. MD is the **easy and cost effective** way to cure your memory corruption ills. **Dr. MD** gives a **complete diagnosis** including the source file and line number of your bug.

- *includes source code*
- *non obtrusive*
- *easy to customize*
- *hardware independent (386 not required)*
- *operating system independent (DOS, UNIX, OS/2...)*
- *small library*

Start your code on the road to recovery—order your copy today!

DR. MD™

NEW version 2.0
$129.00 plus shipping

RSS Engineering
Custom and Contract
Software Solutions

2683 N. Taft
Loveland, CO 80538 USA
(303) 669-3300
FAX (303) 669-3354

MasterCard VISA

RSS Engineering
a division of
Remote Switch Systems, Inc.

Circle 32 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

system that must be on at all times to answer telephone calls.

As noted above, DOS has no way of "knowing" how many midnights have passed since the last time the flag was checked when it sees the after-midnight flag byte returned by the BIOS **read time** function. Thus, even with GARLIC installed, if two or more midnights have passed without a DOS-generated **read time** interrupt, one or more days will be lost from the DOS clock.

If you leave your system running directly under DOS 3.3 or later (with GARLIC installed), you will have no problem as long as there is some COMMAND.COM idle time every day. Things are not so simple if you are running under a shell program that inhibits the COMMAND.COM idle-time **read time** cycles. DESQview has this effect, as does any command shell that consists of a continually running batch file. Other shells are probably equally dangerous. I have never run Windows, but I have my suspicions.

If you use a shell program, you can test it (at least in a DOS window) by running the TSR generated by BINGO.ASM (Listing 3) and then running the memory-restoring DEBING.ASM (Listing 4). The procedure is as follows.

Run BINGO, followed by DEBING, both from the default directory. If you get a result similar to that shown in Figure 2 (the offset **Bingo** message one line below and following the returned DOS prompt), there has been a COMMAND.COM-generated **read time** after BINGO has terminated. (Running DEBING is not really necessary in this case, but I show the screen simulation anyway.) If you are running on a laptop with a battery-saving "sleep" mode, wait a few seconds before you try DEBING. When my old PC Convertible is in sleep mode (say, while waiting for keyboard input), DOS pumps out its idle-cycle **read time** command about once every 12 seconds.

However, if you get a result similar to that shown in Figure 3, where the **Bingo** doesn't appear until after DEBING is started, your shell inhibits the COMMAND.COM **read time** checks.

If your system is running unattended all weekend under a shell that suppresses the COMMAND.COM **read time** cycle, you are likely to suffer from the two-midnight vampire (even with GARLIC installed) unless some program containing a DOS time-stamp function runs at least once per day. (An automatic DOS time stamp occurs every time a program asks DOS to create a new file or write to an existing file.) If you are running under DESQview or some other preemptive multitasking environment, you can prevent the two-midnight vampire effect by running a small background program that does nothing but execute the DOS **get time** function (AH=2Ch) and then release its time slice. If you are running under some less accommodating shell that inhibits the DOS automatic **read time** cycle, you are on your own.

FIGURE 2: BINGO under DOS 5.0

```
F:\>BINGO

F:\>
    Bingo
         DEBING

Bingo is gone
F:\>
```

FIGURE 3: BINGO under DESQview

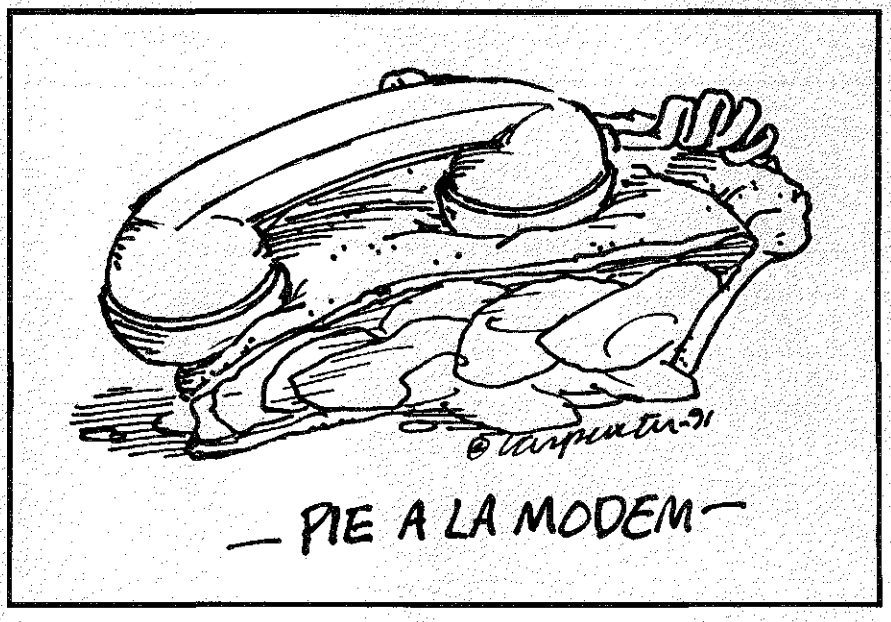
```
F:\>BINGO

F:\>DEBING

Bingo
     Bingo is gone
F:\>
```

DON'T BLAME IT ON DOS

In addition to providing the fix for a long-standing "DOS" problem, this article has a moral: use a little skepticism the next time the power-user press complains about a DOS-induced problem. The "experts" are too willing to blame DOS for all system ills, including those caused by hardware limitations or by the programs running under DOS. (The most famous of these misnamed ills is the 640K "DOS memory" limitation; it is actually a hardware architecture limitation that DOS lives with.) I believe these "experts" are responsible for the undesirable DOS "enhancements" designed to solve imaginary problems—such as tying the two clocks together in DOS 3.3 and providing a substitute CLOCK$ driver in DOS 5.0. We DOS users would be better off if Microsoft (and IBM) stopped listening to those who are too hasty to blame it on DOS. ◘

REFERENCES

1. *BYTE*, July 1991, pp. 303–304.
2. Lesser, Murray L. (1990). "Computing time in C." *Programmer's Journal* 8(1), pp. 75–81.



PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
;                                VAMPIRE.ASM
;    VAMPIRE.COM is a demonstration program to provoke the midnight
; vampire.  (See text for test procedure.)
;                Written by M. L. Lesser, July 25, 1991
;     Assembled with TASM 2.5; linked with TLINK 4.0, switch "/t"
;

CODE    SEGMENT PARA PUBLIC 'CODE'
        ASSUME CS:CODE, DS:CODE

        ORG     100H
VAMPIRE PROC NEAR
        JMP SHORT START

STARTIME DW     2 DUP(?)
MESSGE_1 DB     13,10,'Starting 100-tick count$'
MESSGE_2 DB     13,10,'Ending 100-tick count$'

START:  MOV     AH,0
        INT     1AH
        MOV     STARTIME,DX             ;Save program starting count.
        MOV     STARTIME+2,CX
; Loop until 100 ticks have passed:
        MOV     AH,9                    ;Display startup message.
        MOV     DX,OFFSET MESSGE_1
        INT     21H
AGAIN:  MOV     AH,0
        INT     1AH
        MOV     AX,STARTIME
        SUB     DX,AX
        MOV     AX,STARTIME+2
        SBB     CX,AX
        JNS     OK                      ;We haven't passed midnight.
        ADD     DX,0B0H                 ;Else increase current count by
        ADC     CX,18H                  ;  24 hours.
OK:     SUB     DX,100                  ;Have we gone 100 ticks?
        SBB     CX,0
        JS      AGAIN                   ;If not, go around again.
        MOV     AH,9
        MOV     DX,OFFSET MESSGE_2
        INT     21H
        INT     20H
VAMPIRE ENDP
CODE    ENDS
        END     VAMPIRE
```

LISTING 2

```
;                                GARLIC.ASM
;    GARLIC.COM is a tiny TSR that circumvents the midnight vampire,
; sometimes known as the "lost-date" problem. GARLIC intercepts BIOS
; INT 1Ah, AH=0 (read time) and returns the removed "after midnight"
; flag if that interrupt was not requested from inside DOS.
;                Written by M. L. Lesser, 7/20/91
; Assembled with Borland TASM 2.5; Linked with TLINK 4.0, switch "/t"
;

CODE    SEGMENT PARA PUBLIC 'CODE'
        ASSUME CS:CODE, DS:CODE, ES:CODE

        ORG     100H
BEGIN:  JMP SHORT INIT

ALIGN   4
OLDHANDLER DD   0                       ;original INT 1Ah vector

NEWHANDLER:                             ;new INT 1Ah handler
        STI
        PUSH    AX
        MOV     AX,DS                   ;Did we come from DOS?
        SUB     AX,0070H                ;  If so, DS will hold 0070h.
        POP     AX
        JNZ     NODOS                   ;If not, check for AH=0.
;
;  NOTE:  "Uncommenting" the four statements just below this note
;         converts GARLIC.ASM to GARLICP.ASM.  The addition removes the
;         forward tie between the DOS clock and the real-time clock.
;
;       cmp     ah,3                    ;AH=3 sets real-time clock time.
;       jz      done
;       cmp     ah,5                    ;AH=5 sets real-time clock date.
;       jz      done
```


INSTALIT™

The Premier Worldwide Installation System℠

for DOS • OS/2 • Microsoft® Windows

INSTALIT™ has all the features you've requested over the last five years, so you won't constantly run into walls. It's capable enough to satisfy the world's most complex installation requirements, but so logical it's a joy to use from the start. Take 90 days to decide we're right or your money back. Use it with all your products with no royalties. Demo available!

• All versions share common readable, flexible, procedural, collapsible, encryptable, script language • Rugged, automatic, transparent error handling • Extensive system, environment and network sensing • Unsurpassed compression performance • Suppressible copyright message • Fully automated disk building with transparent huge file splitting • Sophisticated file queueing system. • Easy, flexible text/binary file modification • Easy pop-up windows/dialog boxes • And much, much more!!

"INSTALIT™ is the finest, most complete installation program . . . made the difficult seem trivial. What a delight. I have easily spent hundreds of hours maintaining, patching, kludging and tweaking our former installation program into coping with yet another program or database variation. No more! I'm FREE!"
— Bill Elliott, Context Software Systems, Burr Ridge, IL

"Excellent product. Easy to write very complex installs in a short period of time. . . .encourages you to try different things."
— Randy Jackson, CIS Technologies, Inc., Tulsa, OK

For DOS - $149
Specify national language
For Windows - $199
With C Source add $100
Multilingual for DOS $249
Speaks 10+ languages
VISA/MC/AMEX/PO/COD accepted
Shipping $5, next day $15

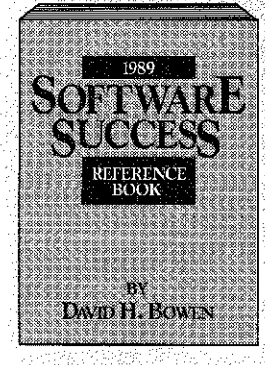
America's Favorite Installer℠
Orders only 1-800-448-4154

HPI

A Product of Helpful Programs, Inc.
P.O. Box 16078 • Huntsville, AL • 35802 • (205) 880-8782 Voice/FAX

Circle 33 on Reader Service Card

THE ANSWER BOOK FOR YOUR SOFTWARE MARKETING PROBLEMS!



SOFTWARE SUCCESS REFERENCE BOOK 1989 by David H. Bowen

I bet at least once today, you've already been faced with a tough decision in some area of your business. Was it Promotion? Lead Generation? Sales? Pricing? A Legal or Management issue? Don't you wish you could pull a book off your shelf and, within seconds have a solution to your problem? Does such a book exist? YOU BET IT DOES!

It's THE SOFTWARE SUCCESS REFERENCE BOOK, a 262-page guide, organized by topic to provide you with FAST HELP in solving your TOUGH PROBLEMS. Compiled from a full year of *SOFTWARE SUCCESS* — the "whole business" newsletter for software CEOs — this indispensable answer book will save you hours of needless worry over making the right decisions for your software company.

Send your check for $50; or call or FAX with your credit card payment (VISA/MC/AEX).

BY PREPAYING, YOU'LL RECEIVE A FREE 3-MONTH SUBSCRIPTION to *SOFTWARE SUCCESS* — a $57 value!

SOFTWARE SUCCESS
P.O. Box 9006
San Jose, CA 95157
Ph: (408) 446-2504
Fax: (408) 255-1098

Circle 34 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
OK:     CLI
        JMP     CS:OLDHANDLER   ;If OK, continue with old interrupt.
NODOS:  OR      AH,AH           ;We are looking only for AH=0 calls.
        JNZ     OK              ;If not AH=0, use old interrupt.
        PUSHF                   ;Simulate INT 1Ah, returning
        CLI                     ;   to this handler.
        CALL    CS:OLDHANDLER
        OR      AL,AL           ;Check "after midnight" flag.
        JNZ     FIXIT           ;If set, replace flag.
DONE:   IRET                    ;Else return to caller.
FIXIT:  PUSH    ES              ;Return flag to BIOS data area.
        PUSH    DI
        XOR     DI,DI           ;Set ES:DI to address of BIOS
        MOV     ES,DI           ; "after midnight" flag.
        MOV     DI,470H
        STOSB                   ; And return flag value.
        POP     DI
        POP     ES
        IRET
ALIGN   16                      ;Transient code starts on paragraph.
;  end of resident code
INIT:   MOV     AX,ES:[2CH]     ;Set ES to segment address of
        MOV     ES,AX           ;  GARLIC's environment block
        MOV     AH,49H          ;  and release memory block.
        INT     21H
        MOV     AX,351AH        ;Save original INT 1Ah vector.
        INT     21H
        MOV     WORD PTR OLDHANDLER,BX
        MOV     WORD PTR OLDHANDLER+2,ES
        MOV     AX,251AH        ; And replace it with NEWHANDLER.
        MOV     DX,OFFSET NEWHANDLER
        INT     21H
;  Terminate and stay resident, freeing memory starting at INIT.
        MOV     DX,OFFSET INIT
        MOV     CL,4
        SHR     DX,CL
        MOV     AX,3100H
        INT     21H
CODE    ENDS
        END     BEGIN
```

LISTING 3

```
;
;                               BINGO.ASM
;   BINGO.COM is a trivial TSR to be used only for checking whether
; DOS is testing for a calendar update during command-line input
; waits. If automatic calendar updating is occurring, running BINGO
; will return the DOS prompt followed by a line feed and the message
; "Bingo"--the cursor will move to the following line (in the column
; following the "o").  If this occurs, BINGO will have removed itself
; from memory.
;   If BINGO is run under a DOS shell that does not support automatic
; calendar updating, only the DOS prompt will return. Remove BINGO
; from memory by immediately running DEBING.
;
;               Written by M. L. Lesser, July 4, 1991
;  Assembled with TASM v 2.5 and linked with TLINK v 4.0, switch "/t"
;

VECTOR  EQU     4*1AH

SHOWIT  MACRO   SYMBOL                  ;Use BIOS to display symbol.
        MOV     AX,0E00H+SYMBOL
        INT     10H
ENDM

CODE SEGMENT PARA PUBLIC 'CODE'
        ASSUME CS:CODE, DS:CODE
        ORG     100H
BINGO   PROC NEAR
        JMP SHORT START         ;initializer
        ALIGN   4
OLDINT  DD      0               ;Save original INT 1AH vector.
NEWINT:         ;new INT 1ah intercept
        OR      AH,AH           ;Check for "read timer."
        JZ      US              ;It's for us.
        JMP     CS:OLDINT       ;Else continue with INT 1AH.
US:     PUSHF                   ;Simulate interrupt,
        CALL    CS:OLDINT       ; returning to next statement.
        STI
        PUSH    BP              ;Insurance, for some video adapters.
        PUSH    AX              ;AX and ES are used in following
        PUSH    ES              ;  portion of NEWINT.
        XOR     AX,AX           ;Restore original interrupt vector.
        MOV     ES,AX
        MOV     AX,WORD PTR CS:OLDINT
        MOV     ES:[VECTOR],AX
        MOV     AX,WORD PTR CS:OLDINT + 2
        MOV     ES:[VECTOR+2],AX
        SHOWIT  0AH             ;;line feed
IRPC    LETTER,Bingo
        SHOWIT  '&LETTER'
ENDM
        SHOWIT  0AH
        MOV     AX,CS           ;Release BINGO's memory.
        MOV     ES,AX
        MOV     AH,49H
        INT     21H
        POP     ES
        POP     AX
        POP     BP
        IRET                    ;Return to COMMAND.COM.

        ALIGN 16
START:                          ;Initialize interrupt intercept.
        MOV     AX,ES:[2CH]     ;environment segment
        MOV     ES,AX
        MOV     AH,49H          ;Release environment memory segment.
        INT     21H
        MOV     AX,351AH        ;Save old INT 1AH vector.
        INT     21H
        MOV     WORD PTR OLDINT,BX
        MOV     WORD PTR OLDINT+2,ES
        MOV     AX,251AH        ;Replace with vector to NEWINT.
        MOV     DX,OFFSET NEWINT
        INT     21H
        MOV     DX,OFFSET START         ;Release memory beyond START.
        MOV     CL,4
        SHR     DX,CL                   ;now in paragraphs
        MOV     AX,3100H
        INT     21H
BINGO   ENDP
CODE    ENDS
        END     BINGO
```

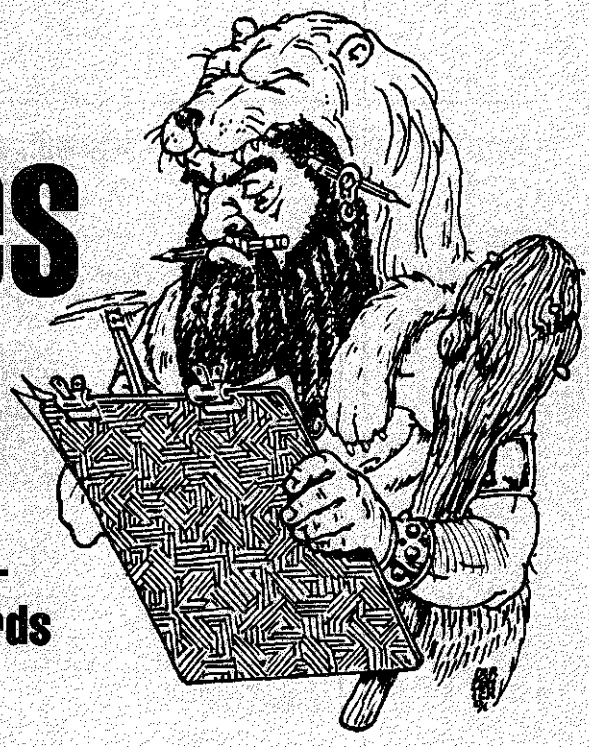
LISTING 4

```
;
;                               DEBING.ASM
;    DEBING removes the BINGO TSR when BINGO has been run in an
; environment that doesn't periodically emit an INT 1AH, AH=0 (such as
; when running in a DOS window under DESQVIEW 386).  If the "Bingo"
; display doesn't follow the DOS prompt after running BINGO, run
; DEBING before running any other program.
;               Written by M. L. Lesser, 7/13/91
;  Assembled with TASM v 2.5 and linked with TLINK v 4.0, switch "/t"
;

SHOWIT  MACRO   SYMBOL                  ;Use BIOS to display symbol.
        MOV     AX,0E00H+SYMBOL
        INT     10H
ENDM

CODE SEGMENT PARA PUBLIC 'CODE'
        ASSUME CS:CODE, DS:CODE
        ORG     100H
DEBING  PROC NEAR
        MOV     AH,0            ;Fire "BINGO."
        INT     1AH
IRPC    LETTER,<Bingo is gone>
        SHOWIT  '&LETTER'
ENDM
        INT     20H
DEBING  ENDP
CODE    ENDS
        END     DEBING
```

All code in this issue is available through our bulletin board services or directly from *PJ* on disk. For information see pages 60 or 77.

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Fast Hercules line drawing



Swifter graphics on the Hercules monochrome and other interleaved video cards

By Nicholas Wilt

The Hercules monochrome graphics standard is old and venerable. It came into existence alongside the IBM Color/Graphics Adapter (CGA) but has almost four times the resolution. CGA and EGA have been eclipsed by VGA and SuperVGA, but Hercules still reigns supreme in monochrome graphics.

To support the Hercules standard, I implemented an eight-octant Bresenham line-drawing algorithm in assembly language for the Hercules adapter. This article concentrates on the code for the Hercules adapter rather than on the line-drawing algorithm. *(For more information about the Bresenham algorithm, see Michael Abrash's "Fast line drawing for the EGA and VGA" in* PJ *7.3. —Ed.)*

The Hercules monochrome standard uses a video memory design that interleaves scan lines in four banks. A similar video memory layout was used on the AT&T PC 6300 mono adapter and also on older Toshiba and Compaq portables. This interleaved memory layout complicates graphics on these adapters.

TAKE IT TO THE BANK

The Hercules has four 8K banks of memory for a total of 32K of display memory. The banks contain four sets of scan lines, as follows:

Bank	Scan lines
0	0, 4, 8, 12, ...
1	1, 5, 9, 13, ...
2	2, 6, 10, 14, ...
3	3, 7, 11, 15, ...

Each scan line is *BytesPerRow* bytes wide, with one bit per pixel. For the standard Hercules resolution of 720 × 348, *BytesPerRow* is 90.

The source code in this article also applies to the AT&T PC 6300 monochrome standard with 640 × 400 resolution. The only difference is that for 640 × 400 resolution, *BytesPerRow* is 80. Thus, the offset into video memory, given an (X,Y) pair, is as follows:

```
Offset = (Y and 3) * 8192
       + (Y / 4) * BytesPerRow + (X / 8)
```

The mask corresponding to an (X,Y) pair is simple to compute:

```
Mask = 128 shr (X and 7)
```

It is more efficient to keep track of offset and mask values while drawing the line than to keep track of *X* and *Y* coordinates and recompute the offset and mask for each point. Thus, you need to update the offset-mask pair to move one pixel in any direction.

Moving to the right is simple: rotate the mask right once and, if the 1-bit in the mask was rotated through the lowermost bit position, increment the index. This is expressed as two instructions in assembly language:

```
ror  Mask,1
adc  Offset,0
```

Moving up or down is more complicated. To move down, follow these steps:

Step 1: *Offset* = *Offset* + 8,192. This moves to the next bank (and, usually, to the next scan line). If this step does not result in integer overflow, you're done.

Step 2: If step 1 resulted in integer overflow, then you have to undo what you just did and compensate:

```
Offset = Offset - 8192     (undo step 1)
Offset = Offset - 24576    (move to bank 1)
Offset = Offset - 7740     (move to the bottom
                             of bank)
```

If *Offset* is negative after this subtraction, you've gone too far. Otherwise, you're done.

Step 3: To undo step 2,

```
Offset = Offset + 7740     (undo step 2)
Offset = Offset + 90       (move down in bank)
```

These steps can be expressed in assembly language as follows:

```
        add  Offset,8192
        jno  Done
        sub  Offset,24576+7740+8192
        jge  Done
        add  Offset,7740+90
Done:   (execution continues)
```

The above code changes *Offset* so that it points to a new location in video memory one scan line below the initial value. A similar code fragment can be developed to move up (see the **DecY** macro in Listing 1). The ability to move one pixel in any direction is important for drawing circles and ellipses as well as for drawing lines.

Nicholas Wilt is a student at Dartmouth College. He can be reached through the offices of Programmer's Journal.

ILLUSTRATION BY JIM CARPENTER

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

LISTING 1

```
; hercline.asm: line-drawing routine for the Hercules monochrome
;      graphics card
; Copyright (C) 1991 by Nicholas Wilt. All rights reserved.

        .MODEL  COMPACT     ; Choose for your memory model.

        .CODE               ; Choose for your memory model.

        ASSUME  cs:_TEXT,ds:_TEXT

        PUBLIC  _hercline

BytesPerRow     EQU     90      ; Change to 80 for AT&T 640x400 mode.
BytesPerBank    EQU     7830    ; Change to 8000 for 640x400 mode.
                                ; BytesPerBank is (YRes/4)*BytesPerRow.

; The IncY and DecY macros update the offset-mask pair in DI and AH to
; point to the above or below pixel. They do not depend on whether
; the routine is drawing or erasing.
IncY    MACRO
        LOCAL   ExitIncY
        add     di,bp           ;; 8192 in BP
        jnc     ExitIncY        ;; Leave if done.
                                ;; Undo and compensate.
        sub     di,24576+BytesPerBank-BytesPerRow+8192
        jge     ExitIncY        ;; Leave if done.
        add     di,BytesPerBank ;; Undo and compensate.
ExitIncY:
        ENDM

DecY    MACRO
        LOCAL   ExitDecY
        sub     di,bp                   ;; 8192 in BP
        jns     ExitDecY                ;; Leave if done.
        sub     di,BytesPerRow-8192-24576 ;; Undo and compensate.
        jge     ExitDecY                ;; Leave if done.
        add     di,BytesPerBank         ;; Undo and compensate.
ExitDecY:
        ENDM

; Void hercline(int x1, int y1, int x2, int y2, int color);
; C-callable in tiny, small, compact memory models.
; To use medium, large, or huge, just add 2 to the indices when
; loading the parameters.
; Color is nonzero to draw the line, 0 to erase the line (set to 0).

hercline_parms struc
                dw    2 dup(?) ; Pushed BP and return address.
    hl_x1       dw    ?
    hl_y1       dw    ?
    hl_x2       dw    ?
    hl_y2       dw    ?
    hl_color    dw    ?
hercline_parms ends
_hercline       PROC
        push    bp              ; Enter.
        mov     bp,sp           ; Set up stack frame.
        push    ds              ; Save regs.
        push    di              ;
        push    si              ;

; Add two to each of the indices below to use a large code model. For
; example, [bp+hl_x1]
        mov     ax,[bp+hl_x1]       ; Load x1.
        mov     bx,[bp+hl_y1]       ; Load y1.
        mov     cx,[bp+hl_x2]       ; Load x2.
        mov     dx,[bp+hl_y2]       ; Load y2.
        mov     bp,[bp+hl_color]    ; Load color into BP. Dangerous.

        cld                     ; String ops go forward.
        push    cs              ; Copy CS to DS for table lookup.
        pop     ds              ;

        sub     dx,bx           ; DX <- diffy
        sub     cx,ax           ; CX <- diffx
        jge     CheckForNull    ; Jump if x2 >= x1.
        add     ax,cx           ; Make ax contain x2.
        add     bx,dx           ; Make bx contain y2.
        neg     dx              ; Negate diffx and diffy.
        neg     cx              ;
CheckForNull:                   ; Make sure the line's start and end
                                ; points are different.
        jnz     DrawLine        ; If diffx nonzero, draw the line.
        or      dx,dx           ; Check diffy against 0.
        jnz     DrawLine        ; If not zero, draw the line.
        jmp     LeaveLine       ; Do not try to draw line if
                                ; x1==x2 && y1==y2.
DrawLine:
        mov     di,ax           ; Load X into DI.
        shr     di,1            ; Divide by 8 for initial offset.
        shr     di,1            ;
        shr     di,1            ;

        push    cx              ; Save DIFFX and DIFFY on stack.
        push    dx              ;
        xchg    ax,cx           ; CX <- X; AX <- DIFFX
        mov     ax,bx           ; AX <- Y

        and     bx,3            ; Add bank into offset.
        shl     bx,1            ;
        add     di,Banks[BX]    ; Lookup table located after routine.

        shr     ax,1            ; Divide Y by 4.
        shr     ax,1            ;
        mov     dx,BytesPerRow  ; Load factor to multiply by.
        mul     dx              ; Perform multiply.
        add     di,ax           ; Add to offset.

        and     cl,7            ; x &= 7
        mov     ah,80h          ; AH <- Initial mask
        shr     ah,cl           ;

        mov     cx,0B000H       ; Point DS and ES at video memory.
        mov     ds,cx           ;
        mov     es,cx           ;
```


PJ Code is available on the following bulletin board services:

The Buffer Board
538 Cardinal Way, Springfield, OR 97477
(Data) 503-747-7636
The Buffer Board contains all of PJ's code from the Sept/Oct 1987 issue through the current issue.

PCS-49 on CompuServe
Section 10, sponsored by: MISOSYS, Inc.
P.O. Box 239, Sterling, VA 22170
(Voice) 703-450-4181

The Silicon Zone
4419 W. Hillsboro Blvd., Suite 290, Coconut Creek, FL 33073
(Data) 305-360-9376

Generation Five Inc.
12061 Tech Road, Silver Spring, MD 20904-9998
(Voice) 301-680-3600 **(Data) 301-680-9300**
Accessible via PC Pursuit, Washington, D.C. From Files submenu, select "P" for Personal Downloads.

The Programmer's Corner
5430F Lynx Lane #224, Columbia, MD 21044
(Data) 301-596-1180, 621-4636, or 995-3744

A.C.M. Phoenix Chapter
6408 East Palm Lane, Scottsdale, AZ 85257-2563
(Voice) 602-990-3102 **(Data) 602-970-0474**

Gilmore Systems of Southern California
1285 Falling Star Avenue, Thousand Oaks, CA 91362
(Voice) 818-706-9800 **(Data) 818-706-9805**

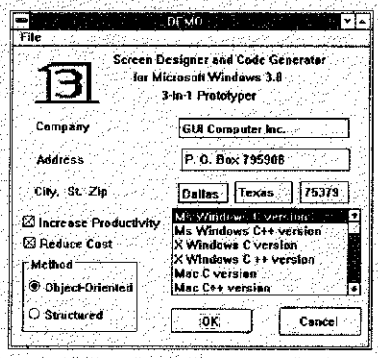
Mnematics Videotex
P.O. Box 19, Sparkill, NY 10976-0019
(Voice) 800-322-3633 (Voice) 914-365-0184 in N.Y.
*Mention **PJ** when you call.*

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C/C++ WINDOWS 3.0
Programming Tool Box

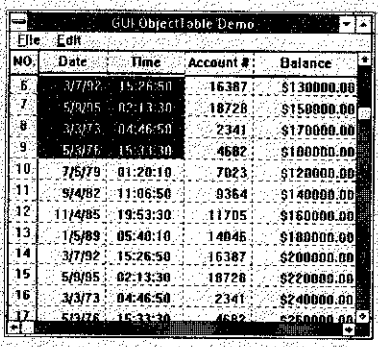
3-in-1 C/C++
Screen Designer & Code Generator

Complete any screen design and generate C/C++ Code, and all the other files in minutes.
C $159 C++ $199 Both $259



ObjectTable C/C++
Source Code Utility

Multi-column, title, row number, auto data validation, editable table. Manage large than 32 K data, ease programming.
C $199 C++ $259 Both $299



DynamicObject C/C++
Source Code of Screen Designer

Dynamically manipulate the objects on the window: create, move, size, cut and paste objects.
C $159 C++ $199 Both $259

GUI Computer Inc.
P.O. Box 795908 Dallas, Texas 75379

1-800-800-9010
TEL 214-250-3472 Fax 214-250-1355

Circle 35 on Reader Service Card

DRAWING THE LINE

Implementing the line-drawing routine is straightforward. It takes two (X,Y) pairs and a color as arguments. The (X,Y) pairs are assumed to be in range (0–719 for X and 0–347 for Y); color is zero or nonzero (nonzero draws, and zero erases).

Listing 1 shows the assembly language source for the Hercules line-drawing routine. The *BytesPerRow* and *BytesPerBank* equates are the only parts that need to be changed to support the AT&T 640 × 400 mode. The code has been optimized with register allocation and code duplication. Nevertheless, the routine can be modified easily for various memory models and calling conventions, because it loads the parameters in a small, contiguous section of code.

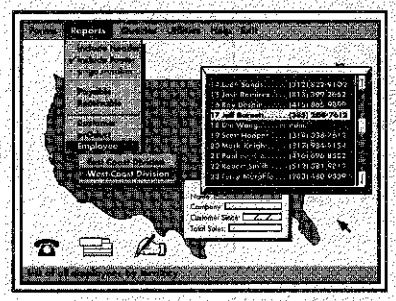
IMPLEMENTATION TESTING

On the code disk for this issue, I have included a demonstration and test program (HERCTEST.C), which is written for Turbo C (Turbo C++ and Borland C++ will also work). It is intended to ferret out problems with the routine and to measure the increase in speed that the assembly language implementation provides. First, the routine draws 1,000 random lines; then the same 1,000 lines are redrawn with Borland's C/C++ library routine. The program reports how much faster this routine is than the Borland C library routine and gives our assembly code another workout by drawing lines in every possible direction. If the test routine has errors, they are usually isolated so that certain types of lines are drawn correctly and other types are drawn incorrectly. If there is a problem with the routine, this final test makes it obvious and indicates the problem's probable location.

Compared with Turbo C++, our assembly language implementation is about four times faster. This speed difference will depend on your machine (80x86) and on the video adapter used (whether it's a "real" Hercules-compatible adapter or an EGA/VGA card with a Hercules emulation).

Although the source code in Listing 1 implements fast line drawing on the Hercules monochrome standard, it can easily be adapted for other interleaved video memory schemes and even simpler linear memory layouts. Many VGA adapters also emulate the Hercules standard, so I suspect we'll be supporting this "older technology" for some time to come. DJ

If You Program
in C, C++, Basic, Fortran, Cobol, Pascal, dBase, *.*
HI-SCREEN Pro II™
is your user interface solution



With HI-SCREEN Pro II, you can design modern text and graphic user interfaces in a snap:

- Design interface objects using interactive editors. Generate all windows, icons, menus, and data entry screens in a WYSIWYG fashion.
- Test all objects directly under the editors, including data validation, menus, icons and dialog boxes.
- Manage objects from your program using a set of versatile and powerful functions.

Hercules, CGA, EGA, VGA, 25/30/43/50 line modes (auto-detect) • Full mouse support & keyboard control • Automatic data validation • Screen editor functions include: view, load, merge, copy, delete, color, draw (using any character), frame (standard, 3D, round corners), undo, on-line ASCII table, merge graphic screens & icons for hypertext effects, direct access to other editors (icon editor, menu editor, graph editor, etc.) • Field types: alphanumeric, real, integer, long text, chosen characters, date, time, list, mouse box, selector, password, flag button • Field description: type, range, format, access key, justification, color, help message or screen, etc. • Data entry test mode under the editor • Automatic context-sensitive help systems • Automatic scrolling windows with scroll bars & boxes • Automatic shadowing effects • Combine text and graphics • Capture any text and graphic screens • Flexible input management: field-by-field or full-screen, interrupt input before or after field, run parallel task, check current field, etc. • Automatic menu systems: pull-down, horizontal, vertical • Link screens into .EXE or keep a separate library file. Only $395 • No royalties • Call for Windows 3 and OS/2 modules.

CALL: 1-800-338-2852

Softway, Inc.
185 Berry Street, Suite 5411, San Francisco, CA 94107
Tel.: 415-896-0708 • Fax: 415-896-0709

Circle 36 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Announcing version 2:

Victor Image Processing Library

Use Victor to develop powerful image applications

Work with images of any size -- use conventional, expanded, and extended memory

Now your applications can support 8-bit color and gray scale images of any size because Victor gives you complete control over conventional, expanded, and extended memory.

Image-based applications can be developed in MSC, QuickC, and Turbo C environments.

Display on Super VGA

Display images on EGA/VGA and super VGA up to 1024 x 768 256 colors.

Load & save PCX/TIFF/GIF/BIN

Handle images from any source, or create translation programs between the popular file formats.

Your application will be able to print halftones at any size.

Gray scale and color images

Powerful image processing for all images -- your software can have features like: zoom, resize, brighten, contrast, sharpen, outline, linearize, matrix conv, colorize, & more.

ScanJet and LaserJet support

You can offer device control for gray scale scanning -- AND print halftones at any size.

Victor supports Microsoft C, QuickC, and TurboC, includes demonstration and prototyping software, and full documentation. Source code available.

Give your applications support for scanner and laser printer.

***Victor Library version 2*, $195**

Call (314) 962-7833 to order VISA/MC/COD

Catenary Systems 470 Belleview St Louis MO 63119 (314) 962-7833

Circle **37** on Reader Service Card

Programmer Power Tools

Introducing: BASH™

A High Performance Key-Indexed Record Management System for DOS

The power of a database without the hassles! Use the BASH tool kit to quickly develop complex, high performance data management applications. BASH functions allow your application to quickly retrieve, insert, update, and delete records. Access records sequentially or randomly using single or multiple keys. You may choose the widely acclaimed B-tree indexing method, a hashed indexing method or both. Record locking is provided for multitasking and LAN applications. The highly optimized BASH kernel uses only 25K and is accessible from most DOS programming languages. BASH really is the best choice when it comes to developing data management applications.

$89 (object code only)

$249 (C source code)

DOS Multitasking with MultiDos Plus™

Elegant, easy to use, powerful DOS multitasking. Discover the product that OEMs around the world use for their critical applications. Thousands of customers and five years of proven reliability make MultiDos Plus the best choice for your multitasking platform.

- *Fast task context switch time*
- *Very flexible prioritized preemptive task scheduling*
- *Small memory overhead (64K)*
- *Complete complement of API functions at no extra charge*
- *Task execution in UM 4.0 EMS lets your application grow without leaving DOS*
- *Excellent for real-time turnkey systems*
- *In-depth documentation with lots of sample software*

$99 (object code only)

Call today for complete information, demo or to order.

(800) 678-2141

(508) 651-0091

FAX (508) 655-8860

BBS (508) 650-9552

Nanosoft Inc.

13 Westfield Road

Natick, MA 01760

VISA MasterCard

Circle **38** on Reader Service Card

```
        mov     al,[di]         ; Pick up initial value.
        pop     dx              ; Restore DIFFX and DIFFY.
        pop     bx              ;
        or      dx,dx           ; If DIFFY 0, draw span.
        jnz     CheckDrawing
        call    hercspan        ; Call span routine.
        jmp     LeaveLine       ; Leave.
CheckDrawing:
        or      bp,bp           ; Are we drawing?
        jnz     CheckDIFFY      ; Yes, don't take one's comp of mask.
        not     ah              ; AH <- ~AH
CheckDIFFY:
        or      dx,dx           ; Compare DIFFY to 0.
        jl      DIFFYNegative   ; Jump if less than.
        jmp     DIFFYPositive
DIFFYNegative:
        mov     cx,dx           ; Copy DIFFY to CX for compare.
        neg     cx              ; Negate for compare.
        cmp     cx,bx           ; Compare to DIFFX.
        jle     LineXYM         ; Jump if -diffy <= diffx.
LineYMX:                        ; Decrement Y before incrementing X
                                ; (count is already in cx).
        mov     si,dx           ; Initialize error.
        sar     si,1            ;
        or      bp,bp           ; Are we drawing?
        mov     bp,8192         ; Set up for changes in Y.
        jne     DrawYMX         ; Jump if yes.

; Loop: erase, decrement Y, and increment X if appropriate.

EraseYMX:
        and     byte ptr [di],ah ; Erase.
        sub     di,bp           ; Decrement Y.
        js      FixupEraseYMX   ; Jump if we need to fix up.
ContinueEraseYMX:
        add     si,bx           ; Add DIFFX to error.
        jns     ELoopYMX        ; Jump if positive.
        loop    EraseYMX        ; Loop until done.
        jmp     short DoneEYMX  ; Leave if done.
FixupEraseYMX:
        sub     di,BytesPerRow-8192-24576 ; Undo and compensate.
        jge     ContinueEraseYMX  ; Leave if done.
        add     di,BytesPerBank   ; Undo and compensate.
        jmp     short ContinueEraseYMX
ELoopYMX:
        ror     ah,1            ; Increment X.
        sbb     di,-1           ;
        add     si,dx           ; Add DIFFY to error.
        loop    EraseYMX        ; Loop until done.
DoneEYMX:
        and     byte ptr [di],ah ; last plot
        jmp     LeaveLine       ; Jump to exit code.

; Loop: draw, decrement Y, and increment X if appropriate.

DrawYMX:
        or      byte ptr [di],ah ; plot
        sub     di,bp           ; Decrement Y.
        js      FixupDrawYMX    ; Jump if fixup needed.
ContinueDrawYMX:
        add     si,bx           ; Add DIFFX to err.
        jns     DLoopYMX        ; Jump if positive.
        loop    DrawYMX         ; Loop until done.
        jmp     short DoneDYMX  ; Jump if done.
FixupDrawYMX:
        sub     di,BytesPerRow-8192-24576 ; Undo and compensate.
        jge     ContinueDrawYMX           ; Leave if done.
        add     di,BytesPerBank           ; Undo and compensate.
        jmp     short ContinueDrawYMX
DLoopYMX:
        ror     ah,1            ; Increment X.
        adc     di,0            ;
        add     si,dx           ; Add DIFFY.
        loop    DrawYMX         ; Loop until done.
DoneDYMX:
        or      byte ptr [di],ah ; last plot
        jmp     LeaveLine       ; Jump to exit code.

LineXYM:
        mov     cx,bx           ; Get count.
        mov     si,bx           ; Initialize error.
        neg     si              ;
        sar     si,1            ;
        or      bp,bp           ; Are we drawing?
        mov     bp,8192         ; Set up for changes in Y.
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

```
        jne     DrawXYM         ; Jump if drawing.

; Loop: erase, increment X, and decrement Y if appropriate.

EraseXYM:
        and     al,ah           ; Erase.
        ror     ah,1            ; Rotate mask right.
        jnc     GetNextEraseXYM ; Jump if we need another byte.
ContinueEraseXYM:
        sub     si,dx           ; Subtract DIFFY from err.
        jns     ELoopXYM        ; Jump if positive.
        loop    EraseXYM        ; Loop until done.
        jmp     short DoneEXYM  ; Jump if done.
GetNextEraseXYM:
        stosb                   ; Store and increment DI.
        mov     al,[di]         ; Get next byte.
        jmp     short ContinueEraseXYM
ELoopXYM:
        mov     [di],al         ; Save to video memory.
        DecY                    ; Decrement Y.
        mov     al,[di]         ; Load from video memory.
        sub     si,bx           ; Subtract DIFFX from err.
        loop    EraseXYM        ; Loop until done.
DoneEXYM:
        and     al,ah           ; Plot last.
        jmp     WriteLastByte   ; Jump to exit code.

; Loop: draw, increment X, and decrement Y if appropriate.

DrawXYM:
        or      al,ah           ; plot
        ror     ah,1            ; Rotate mask right.
        jc      GetNextDrawXYM  ; Jump if we need another byte.
DrawContinueXYM:
        sub     si,dx           ; Subtract DIFFY from err.
        jns     DLoopXYM        ; Jump if positive.
        loop    DrawXYM         ; Loop until done.
        jmp     short DoneDXYM  ; Jump if done.
GetNextDrawXYM:
        stosb                   ; Store and increment DI.
        mov     al,[di]         ;
        jmp     short DrawContinueXYM
DLoopXYM:
        mov     [di],al         ; Save to video memory.
        DecY                    ; Decrement Y.
        mov     al,[di]         ; Load from video memory.
        sub     si,bx           ; Subtract DIFFX.
        loop    DrawXYM         ; Loop until done.
DoneDXYM:
        or      al,ah           ; Plot last.
        jmp     WriteLastByte   ; Jump to exit code.

DIFFYPositive:
        cmp     dx,bx           ; Compare DIFFY to DIFFX.
        jle     LineXYP         ; Jump if less than or equal to.
LineYPX:
        mov     cx,dx           ; Load count.
        mov     si,dx           ; Initialize error.
        neg     si              ;
        sar     si,1            ;
        or      bp,bp           ; Are we drawing?
        mov     bp,8192         ; Set up for changes in Y.
        jne     DrawYPX         ; Jump if yes.

; Loop: erase, increment Y, and increment X if appropriate.

EraseYPX:
        and     byte ptr [di],ah ; Erase.
        add     di,bp           ; Increment Y.
        jo      FixupEraseYPX   ; Jump if fixup needed.
ContinueEraseYPX:
        add     si,bx           ; Add DIFFX to err.
        jns     ELoopYPX        ; Jump if positive.
        loop    EraseYPX        ; Loop until done.
        jmp     short DoneEYPX  ; Jump if done.
FixupEraseYPX:
        sub     di,24576+BytesPerBank-BytesPerRow+8192
        jge     ContinueEraseYPX ; Leave if done.
        add     di,BytesPerBank ; Undo and compensate.
        jmp     short ContinueEraseYPX
ELoopYPX:
        ror     ah,1            ; Increment X.
        sbb     di,-1           ;
        sub     si,dx           ; Subtract DIFFY from err.
        loop    EraseYPX        ; Loop until done.
```


Passive Backplane CPUs

PC Tech manufactures a full line of modular passive backplane high performance PC compatible CPUs.

Passive backplane based systems have many advantages, including ease of assembly, flexibility, fast service (by swapping boards), and simple upgrades to new or faster processors.

Contact us for more information, pricing, and benchmark results.

80386SX: 20 MHz, 512K through 16M DRAM
80386DX: 25 MHz & 33 MHz, 1M - 32M DRAM, 128K cache module optional
80486: 25 MHz & 33 MHz, 1M - 32M DRAM, 256K cache module optional

Call, write, or FAX for complete specifications and retail or OEM terms. All PC Tech products are available in complete systems as well as components for retail, OEM, and private label sales.

Designed, Manufactured, Sold and Serviced by:

PC Tech Inc.
Voice . . . (612) 345-4555
FAX . . . (612) 345-5514
Modem . . (612) 345-4656

907 N. 6th St., Lake City, MN 55041

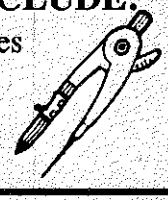
Circle 39 on Reader Service Card

ProGraphx Toolbox™

New Version 2.0 Write for a Free Demo Disk

NEW FEATURES INCLUDE:

- 320x200 EGA mode for games
- Icon Animation Editor
- Great for games, real-time,
- or detail-oriented graphics



Over 50 highspeed graphics routines for EGA/VGA 16 color modes, including proprietary color icon routines. **ProGraphx Image Editor** is a complete color icon paint package which allows you to edit and view up to 50 icons at once. Simply paint an icon, then use the **ProGraphx Toolbox™** routines to display it with amazing speed. Over 500 sample icons for use in your own programs. Includes **Operation: Airlift**, a full-screen scrolling arcade style game with source code. Works with **ANY** C or Turbo Pascal.

All routines written in assembly forfastest performance

SPECIAL OFFER
$99 with source
No Royalties!

To order, write:
Peder Jungck
1019 West Lovers Lane
Arlington, TX 76013
Include $5 S&H

Circle 40 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Imagine...

Information on Demand

Imagine having the information you want at your fingertips. Imagine that it's free.

Programmer's Journal Reader Service Cards—where imagination and reality come together.

Valuable information on products and services can be yours **FREE**—we even pay the postage! Just tear out the ***PJ*** Reader Service Card opposite the Ad Index on page 76. Circle the numbers from the ads or editorial mentions of the items you want to know more about.

Now, imagine yourself filling out the card and sending it in . . . today.

U.S. Postal Service

STATEMENT OF OWNERSHIP, MANAGEMENT AND CIRCULATION
Required by 39 U.S.C. 3685)

1A. TITLE OF PUBLICATION	1B. PUBLICATION NO.	2. DATE OF FILING
Programmer's Journal	0 0 0 2 9 3	10-1-91
3. FREQUENCY OF ISSUE	3A. NO. OF ISSUES PUBLISHED ANNUALLY	3B. ANNUAL SUBSCRIPTION PRICE
Bimonthly	6	$29.70

4. COMPLETE MAILING ADDRESS OF KNOWN OFFICE OF PUBLICATION *(Street, City, County, State and ZIP+4 Code) (Not printer)*

150 N 4th Springfield OR 97477-5454

5. COMPLETE MAILING ADDRESS OF THE HEADQUARTERS OF GENERAL BUSINESS OFFICES OF THE PUBLISHER *(Not printer)*

PO Box 70167 Eugene OR 97401-0110

6. FULL NAMES AND COMPLETE MAILING ADDRESS OF PUBLISHER, EDITOR, AND MANAGING EDITOR *(This item MUST NOT be blank)*

PUBLISHER *(Name and Complete Mailing Address)*

Elizabeth Oakley PO Box 70167 Eugene OR 97401-0110

EDITOR *(Name and Complete Mailing Address)*

M. Steven Baker PO Box 70167 Eugene OR 97401-0110

MANAGING EDITOR *(Name and Complete Mailing Address)*

Cecelia Hagen PO Box 70167 Eugene OR 97401-0110

7. OWNER *(If owned by a corporation, its name and address must be stated and also immediately thereunder the names and addresses of stockholders owning or holding 1 percent or more of total amount of stock. If not owned by a corporation, the names and addresses of the individual owners must be given. If owned by a partnership or other unincorporated firm, its name and address, as well as that of each individual must be given. If the publication is published by a nonprofit organization, its name and address must be stated.) (Item must be completed.)*

FULL NAME	COMPLETE MAILING ADDRESS
Oakley Publishing Company	PO Box 70167 Eugene OR 97401-0110
Elizabeth Oakley	PO Box 70167 Eugene OR 97401-0110
James Oakley	PO Box 70167 Eugene OR 97401-0110

8. KNOWN BONDHOLDERS, MORTGAGEES, AND OTHER SECURITY HOLDERS OWNING OR HOLDING 1 PERCENT OR MORE OF TOTAL AMOUNT OF BONDS, MORTGAGES OR OTHER SECURITIES *(If there are none, so state)*

FULL NAME	COMPLETE MAILING ADDRESS
None	

9. FOR COMPLETION BY NONPROFIT ORGANIZATIONS AUTHORIZED TO MAIL AT SPECIAL RATES *(Section 423.12 DMM only)*
The purpose, function, and nonprofit status of this organization and the exempt status for Federal income tax purposes *(Check one)*

(1) ☐ HAS NOT CHANGED DURING PRECEDING 12 MONTHS
(2) ☐ HAS CHANGED DURING PRECEDING 12 MONTHS *(If changed, publisher must submit explanation of change with this statement.)*

10. EXTENT AND NATURE OF CIRCULATION *(See instructions on reverse side)*	AVERAGE NO. COPIES EACH ISSUE DURING PRECEDING 12 MONTHS	ACTUAL NO. COPIES OF SINGLE ISSUE PUBLISHED NEAREST TO FILING DATE
A. TOTAL NO. COPIES *(Net Press Run)*	30,022	27,270
B. PAID AND/OR REQUESTED CIRCULATION 1. Sales through dealers and carriers, street vendors and counter sales	12,419	12,826
2. Mail Subscription *(Paid and/or requested)*	15,190	11,385
C. TOTAL PAID AND/OR REQUESTED CIRCULATION *(Sum of 10B1 and 10B2)*	27,609	24,211
D. FREE DISTRIBUTION BY MAIL, CARRIER OR OTHER MEANS SAMPLES, COMPLIMENTARY, AND OTHER FREE COPIES	1,079	587
E. TOTAL DISTRIBUTION *(Sum of C and D)*	28,688	24,798
F. COPIES NOT DISTRIBUTED 1. Office use, left over, unaccounted, spoiled after printing	5,030	2,472
2. Return from News Agents	3,716	0
G. TOTAL *(Sum of E, F1 and 2—should equal net press run shown in A)*	30,022	27,270

11. I certify that the statements made by me above are correct and complete

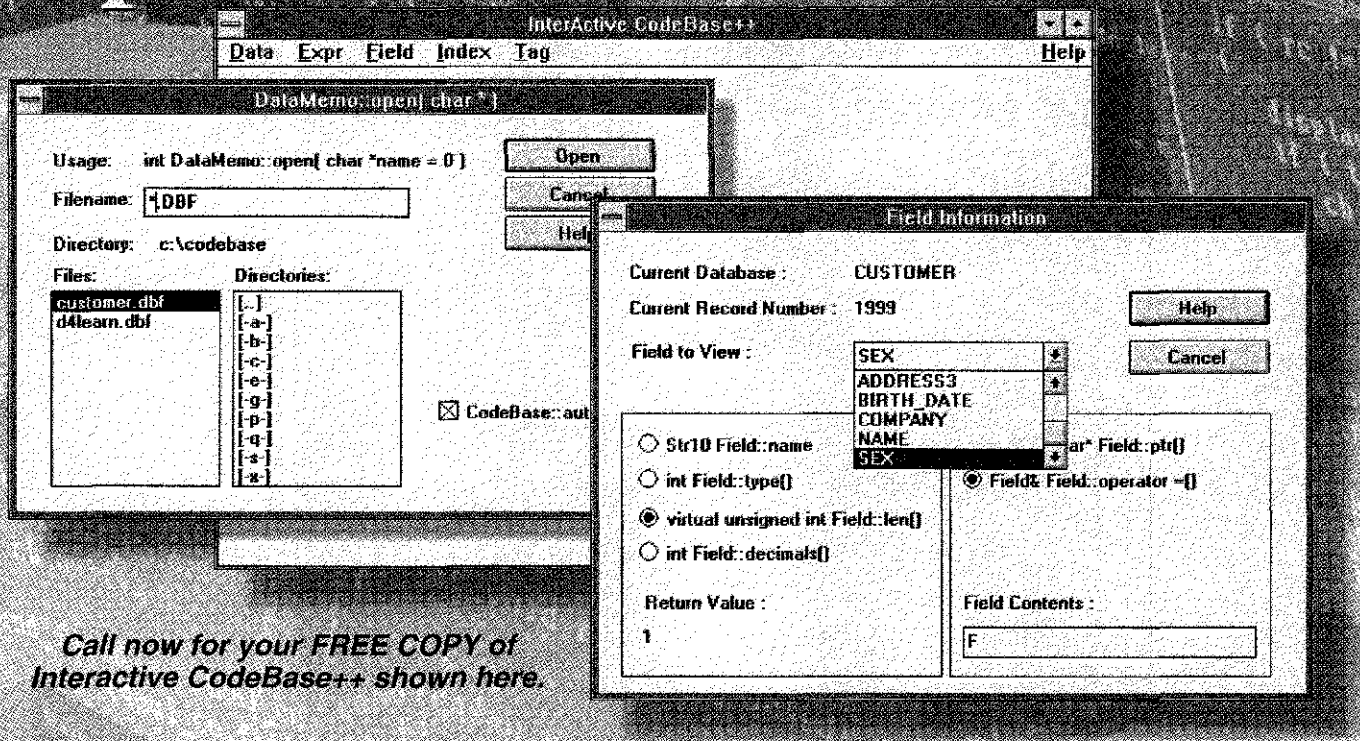
SIGNATURE AND TITLE OF EDITOR, PUBLISHER, BUSINESS MANAGER, OR OWNER
Elizabeth Oakley

PS Form 3526, Dec. 1985 *(See Instructions on reverse)*

```
DoneEYPX:
        and     byte ptr [di],ah ; Erase last.
        jmp     LeaveLine        ; Jump to exit code.

; Loop: draw, increment Y, and increment X if appropriate.

DrawYPX:
        or      byte ptr [di],ah ; plot
        add     di,bp            ; Increment Y
        jo      FixupDrawYPX     ; Jump if fixup needed.
ContinueDrawYPX:
        add     si,bx            ; Add DIFFX to err.
        jns     DLoopYPX         ; Jump if positive.
        loop    DrawYPX          ; Loop until done.
        jmp     short DoneDYPX   ; Jump if done.
FixupDrawYPX:
        sub     di,24576+BytesPerBank-BytesPerRow+8192
        jge     ContinueDrawYPX  ; Leave if done.
        add     di,BytesPerBank  ; Undo and compensate.
        jmp     short ContinueDrawYPX
DLoopYPX:
        ror     ah,1             ; Increment X.
        adc     di,0             ;
        sub     si,dx            ; Subtract DIFFY from err.
        loop    DrawYPX          ; Loop until done.
DoneDYPX:
        or      byte ptr [di],ah ; Plot last.
        jmp     short LeaveLine  ; Jump to exit code.

LineXYP:
        mov     cx,bx            ; Load count.
        mov     si,cx            ; Initialize error.
        neg     si               ;
        sar     si,1             ;
        or      bp,bp            ; Are we drawing?
        mov     bp,8192          ; Set up for changes in Y.
        jne     DrawXYP          ; Jump if yes.

; Loop: erase, increment X, and increment Y if appropriate.

EraseXYP:
        and     al,ah            ; Erase.
        ror     ah,1             ; Rotate mask right.
        jnc     GetNextEraseXYP  ; Jump if we need another byte.
ContinueEraseXYP:
        add     si,dx            ; Add DIFFY to err.
        jns     ELoopXYP         ; Jump if positive.
        loop    EraseXYP         ; Loop until done.
        jmp     short DoneEXYP   ; Jump to end.
GetNextEraseXYP:
        stosb                    ; Store and increment DI.
        mov     al,[di]          ; Get next byte.
        jmp     short ContinueEraseXYP
ELoopXYP:
        mov     [di],al          ; Save to video memory.
        IncY                     ; Increment Y.
        mov     al,[di]          ; Load from video memory.
        sub     si,bx            ; Add DIFFX to err.
        loop    EraseXYP         ; Loop until done.
DoneEXYP:
        and     al,ah            ; Erase last.
        jmp     short WriteLastByte

; Loop: draw, increment X, and increment Y if appropriate.

DrawXYP:
        or      al,ah            ; plot
        ror     ah,1             ; Rotate mask right.
        jc      GetNextDrawXYP   ; Jump if we need another byte.
DrawContinueXYP:
        add     si,dx            ; Add DIFFY to err.
        jns     DLoopXYP         ; Jump if positive.
        loop    DrawXYP          ; Loop until done.
        jmp     short DoneDXYP   ; Jump if done.
GetNextDrawXYP:
        stosb                    ; Store byte and increment.
        mov     al,[di]          ; Get next byte.
        jmp     short DrawContinueXYP
DLoopXYP:
        mov     [di],al          ; Save to video memory.
        IncY                     ; Increment Y.
        mov     al,[di]          ; Load from video memory.
        sub     si,bx            ; Subtract DIFFX from err.
        loop    DrawXYP          ; Loop until done.
DoneDXYP:
```


PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Why do dBASE professionals use C++?



Call now for your FREE COPY of Interactive CodeBase++ shown here.

• dBASE Functionality

Use CodeBase++, a C++ class library, for database management.

• Object Oriented Syntax

dBASE Code

```
* Assign to field LAST_NAME
LAST_NAME = "Jones"
```

CodeBase++ Code

```
// Assign to field LAST_NAME
LAST_NAME = "Jones" ;
```

• Low Memory Requirements

A CodeBase++ .EXE file is as small as 20k.

• Speed

"CodeBase++ is fast: I indexed a .dbf of 10,000 records in 32 seconds; dBASE IV took 40 seconds." Kathy Cea, DBMS

• Portability

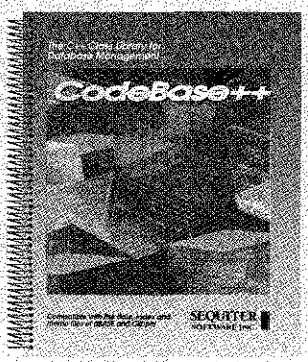
CodeBase++ ships with a Microsoft Windows 3.0 DLL.

• Compatibility

Codebase++ is completely compatible with dBASE and Clipper data and index files.

CodeBase++

The Power of dBASE, the Versatility of C++



SEQUITER SOFTWARE INC.

TEL. 403•437•2410
FAX 403•436•2999

#209,9644-54 AVE., EDMONTON, AB, CANADA T6E-5V1

Circle 54 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

THE NEW STANDARD IN EXCELLENCE THE NEW STANDARD IN EXCELLENCE THE NEW STANDARD IN EXCELLENCE

Keep on Truking

By Dave Thompson

I stepped off the plane into 95-degree heat and 98-percent humidity. I was in a foreign country somewhere in the middle of the Pacific. I didn't know the language, I didn't know the customs (or the Customs agent), and I'd never met Alan, the person who was supposed to meet me.

As the plane prepared to leave for the next leg of its trip, an agent came over to ask me if I really wanted to stay. I wasn't sure. People had pretty well cleared the airport—an open, cement-block shed surrounded by a chain-link fence—and Alan hadn't showed. I had five bags totaling 200 pounds and no way to get them out of the airport. Even if transportation had been available, I didn't have the slightest idea where to take my stuff. The fence around the airport would soon be locked and remain locked until the plane returned from Guam. There I was—in Truk, Micronesia—to fix computer systems.

THE PLOT THICKENS

Swallowing hard, I chose to stay. I couldn't leave my stuff, which the local kids were eyeing. They wouldn't have known what to do with constant-voltage transformers and uninterruptible power supplies, but I did, and I wanted a chance to do it.

As it turned out, one of the police officers who had airport duty that afternoon not only spoke English but also had once lived in Bend, Oregon—*my* Bend, Oregon. We killed an hour or so talking about the college, favorite restaurants, and skiing. Then he suggested that his brother (the customs agent/immigration officer) watch my gear while I walked the couple of blocks to the town's center of commerce: an oblong, peeling concrete building that housed a motel, a grocery, a department store, and two restaurants, all owned by his cousin. (Getting the picture?)

ILLUSTRATION BY JIM CARPENTER

In the department store I asked whether anyone knew Alan. Everyone stared at me strangely. I remembered that stare from the airport officials. I assumed the problem was language, but I was soon to discover that the problem was Alan.

He turned up half an hour later. (The airport cop I'd been talking to had snagged Alan as he walked past the airport.) Alan, an American, can best be described as an escapee from Haight Ashbury. He was the local computer guru, employed by the high school to keep its 15 computers running.

TROPICAL ELECTRICITY

From the long and rambling correspondence between Alan and MicroSphere (which had sold the systems to the high school), I was guessing that power glitches had fried all the supplies. (Curiously, the only machine still running was Alan's personal system, and he'd been listed persona non grata by the computer science teacher.) The island's 110V power was supplied by a diesel generator feeding a rat's nest of power lines strung on thin, wobbly poles. Storms had been downing power lines with discouraging regularity, and the generator itself had been going down two or three times a week. Add to that the voltage and frequency fluctuations and you have power from hell.

So I'd brought the heavy metal I needed to deal with power problems. Constant-voltage transformers are wonderful for eliminating over-voltage and spike problems. They saturate at 90V, so the output remains at 130V, no matter how much the input rises above 90V.

If the input drops below 90V the output also drops, but it's still a respectable 115V or so at 80V input. The only thing the CV transformer can't deal with is a significant brownout or a blackout. That's why I also stuck an uninterruptible power supply in the line. That way the power would always be covered. The only thing that would get through would be frequency variations, and because PC power supplies rectify the AC into DC before generating the 5V, –5V, and 12V, they're virtually immune to frequency problems.

BEEP ME UP, SCOTTY

So there I was, ready to do combat with power from hell. I fired up the first machine. No display, but a *beep, beep, beep.* I tried another. *Beep, beep, beep.* And another and another. Clearly, those beeping systems didn't have fried power supplies. They'd obviously survived power from hell. The power on self test (POST) was running, but not finishing. The first test is the kernel (the processor, system ROM, and beeper). Then the test checks the system RAM. If the first bank of RAM checks out, the system starts counting RAM on the screen. Otherwise, it just beeps. All 15 systems beeped.

What I hadn't anticipated was corrosion from hell. It turned out that the heat and salty air had corroded the tinning on the dynamic RAM chips. Alan's computer still worked simply because it had a different brand of RAM chips.

I assumed the RAM sockets were at the root of the problem and started unsoldering the plastic nasties from one of the boards. You don't know what hell is until you've tried unsoldering 36 plastic sockets with a pencil iron and a solder sucker. I figured once I got the sockets out, I could solder in the RAM chips. Sure, I'd risk soldering in a bad RAM, but

Dave Thompson is a former hardware/software engineer at Tektronix and the past editor of the prehistoric programming magazine Micro Cornucopia. *He can be reached at P.O. Box 223, Bend, OR 97709.*

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

_miniEd Editing Toolkit
The maximum solution for add-in text editing

Add sophisticated editing capabilities to your applications without costly code overhead or complicated interface procedures. The _miniEd toolkit provides the answer with a set of tools for 'C', BASIC and PASCAL applications. The toolkit includes 2 complete editors to link with your applications :

_miniEdD - Word processing grade editing in a 12K package
• Full Screen or Window Display // • Search & Replace operations
• Insert & Overstrike mode // • Embedded print control characters
• Disk or RAM based editing // • Block operations // • Undo
• Program configurable keystroke and color usage
• Auto reformatting on every keystroke // • AND MORE

_miniEd - RAM based window editor in a 5K package
• Same interface as _miniEdD // • TSRable and ROMable
• Ideal for palmtop & RAM-intensive uses // • Max 64K text size
• Non-destructive VIEW mode for formatted message displays

For developers who need to produce customized editing functions, the tool library also provides independent access to editing sub-routines for :
• Text formatting // • Screen I/O // • Search & Replace // • Disk I/O

The Toolkit is written entirely in 8086 assembler. Full, commented source code is provided for all of the tool library routines. A comprehensive user manual explains how to use all of the Toolkit features. The required interface structures are provided along with sample programs that illustrate tool usage. Call or FAX for a detail sheet and sample listings. Ask about...the Toolkit written in 'C' for applications where portability is paramount.

VISA and MasterCard accepted.
DOS Toolkit-$189.00/Windows 3.0 Toolkit-$279.00/_miniEd-$99.00
Strategic Software Designs, Inc.
Software Tools and Custom Development
6 S 235 Steeple Run Drive - Suite 12B
Naperville, IL 60540
Phone:708/778-6060 FAX:708/778-6063

Circle 42 on Reader Service Card

if there weren't any RAM sockets, there shouldn't be any RAM socket trouble.

THAT KIND OF A DAY

Unfortunately, the unsoldering wasn't working, the sockets were melting, the pads were lifting, and the tip on my solder sucker was cracking. So, after two days, I gave up. I packed the motherboards and the 200 pounds of power conditioning gear I'd brought and headed for the airport.

The plane's schedule was one flight a day. First it flew eastbound along a string of atolls, eventually ending up in Honolulu; the next day it headed west toward Guam. Either way (assuming it didn't break down somewhere) it would get me off Truk. I was sick of the heat, the humidity, the small brown piles of human waste dotting the beaches, the flies, the cholera-contaminated water, the endless Cokes (canned pop and beer were the primary sources of potable liquids), and the diet of Spam and Calrose rice.

I distinctly remember the feeling I had when they pulled up the stairway and fired up the plane's air conditioning. Whew.

When I got back home, MicroSphere checked out the boards and found out that the leads on the RAM chips had corroded. When the chips were replaced, the boards came up. I had trained three of the school's staff to replace the boards. Whether they're successful, we'll see. I may have to go back.

TECHTIPS

I haven't yet received any techtips, but I have seen the T-shirt. The only way you'll get one is to have a techtip printed, and your best shot at getting printed is right now. Jot down one to four paragraphs (and a schematic or short listing if needed) and send them to Techtips, c/o *Programmer's Journal*, P.O. Box 70167, Eugene, OR 97401-0110. And we'll get the whole thing started.

THE Audio Solution

THE Standard for Digitized PC Sound

Add the power of digitized sound to your DOS applications. Use sound to enhance entertainment products, multimedia applications, user interfaces, on-line help systems, and even voice E-Mail.

Supports all 3rd party hardware sound boards, as well as the internal PC speaker with high quality digitized sound. Provides audio compression and decompression utilities with source code. Loadable driver approach provides a simple programmer's interface to produce clean, clear, digitized sound on all hardware platforms.

Sound Board	Operation	Resolution
Covox Speech Thing	BACKGND	8 bit
Covox Voice Master	BACKGND	8 bit
Covox Sound Master	BACKGND	8 bit
Creative Labs Sound Blaster	BACKGND	8 bit/DMA
Adlib Personal Music System	BACKGND	8 bit
Digispeech Adapter Board	FOREGND	8 bit
Walt Disney Sound Source	BACKGND	8 bit
Street Electronics Echo II	BACKGND	8 bit
Lantastic Voice Adapter	BACKGND	8 bit
Tandy EX/TX, PCjr	BACKGND	4 bit
Tandy SL/TL	BACKGND	8 bit/DMA
Internal PC Speaker	both	6 bit

THE Audio Solution, Inc. ♦ 314-567-0267
P.O. Box 11688 ♦ Clayton, MO 63105

The Digitized Sound Package includes:

☞ 12 sound drivers, all with resident size under 1K.

☞ DOS tools to PLAY, MIX, COMPRESS, and DECOMPRESS digitized sound samples.

☞ Wrimage, an entertaining and educational word game that supports up to four players, full of humorous sound effects.

☞ 75 free sound effects!

VISA or MasterCard accepted **$49**

Also available,
FatClips Audio disks
#1 - Sample Music
#2 - Sound Effects
$15 each
$20 for both

FatClips Audio disks are a collection of sampled music and digitized sound effects created by The FatMan, George Sanger, producer of the music for Origin Systems "Wing Commander", and MT-32 arrangements for Lucasfilm Games' "Loom".

***The Digitized Sound Package** was developed by computer game author **John W. Ratcliff** and is in use by Electronic Arts, Strategic Simulations Inc., Milliken Publishing Co., Commodore Computer Inc., and Beyond Software. Reasonable license fees for commercial use.*

Circle 43 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

C++ for big projects

By M. Steven Baker

Hidden inside Balboa Naval Hospital in San Diego is a maze of tunnels. The low-ceilinged hallways house a fleet of robotic vehicles that transport supplies—from cotton swabs to scalpels—throughout Balboa, the largest military hospital in the world. This complex supply-on-demand process begins in the basement inventory center, where workers get requests and punch them into the inventory-control system. Carousels fetch items from inventory and bring them to the workers. After that, robots carry the supplies to nurses' stations throughout the building.

The 300,000 lines of code that control Balboa Hospital's inventory system were written by Intek Integration Technologies, a Bellevue, Washington–based firm that specializes in software for manufacturing and inventory control. The project, composed of several cooperative applications that were written over a period of six months, was created in C and C++ using an interpretive embedded database language.

Intek cofounder Mac Cutchins attributes much of the success of the four-year-old company, which specializes in large-scale software development projects, to its use of C++. The reusability of the language allows the company to compete with larger firms and produce code better and faster. "In many respects," he says, "C++ is the reason we're here."

GO WEST, YOUNG MAC

Mac has been involved in this specialty market since 1982, when he worked with IBM in Boca Raton on the original PC manufacturing line. He started out using BASIC and Fortran.

In 1984, Mac moved to Seattle to work directly with customers on automation projects. Three years later, the firm he was working for decided to standardize on C—a big commitment at the time. With partner Mark Fernandez, Mac founded Intek in late 1987.

The Intek development team devised an interpretive program to handle parts and inventory database requirements for embedded systems. It occurred to them that this part of their real-time system could be rewritten as an object-oriented application.

ILLUSTRATION BY JIM CARPENTER

The company purchased an early version of the Guidelines C++ translator but ran into programming problems—especially memory limitations, despite the much-touted capacity of C++ for handling big programs. In spite of the hurdles, this first exposure gave Cutchins a taste of what could be done with C++.

How difficult was it to switch to C++? Intek consists of 15 people, three of whom (including Mac) are "tools culture" types working in object-oriented programming. Other developers in the company are of the "applications culture" type. The tools programmers build things, and the applications people use them. "We hide details from them—they don't need an OOP point of view or need knowledge of C++," Mac notes, "but a few of us have bitten the learning curve." The switch took a year, and the following year Intek decided to build classes in C++.

Although Intek is a small company working on a "two-tiered" system, Mac considers the benefits of C++ great enough to both individual and team workers to be worth biting the curve. He's written entire applications by himself using C++, including some work for Borland. Mac appreciates the protection, maintenance, and reusability that C++ provides for team projects and values "the ability to retain equity in what's been built."

THE LUCK OF THE DRAW

Mac admits to getting into the C++ compiler business by chance. The company applied to AT&T to license C++ as a way of circumventing the memory problems. 386 DOS extenders were becoming available at about the same time. By early 1988, Intek had licensed AT&T's C++ translator and purchased Phar Lap's DOS extender. Within a few months the development team had redone Intek's database language, embedding it in C++. They liked the results so much that they decided to do future work with their own C++ translator.

"Using the 386 DOS extender, our memory problems disappeared," Mac says. "At last we could use 200K source files." They used the MetaWare High C 386 compiler to compile the output from its CFRONT (C++) translator.

They weren't planning to market and develop their own C++ product, but they started receiving requests for the Intek 386 version of CFRONT 1.2 as soon as word about it got around. "We were just dabbling," Mac says, "but evidently we had built a better mousetrap."

WHY C++ FOR THE XXL ONES

What initially attracted Mac to C++ was the reference operator: it provided a way to globalize database interpreter code, database records, and buffers. "Our interest was in an improved C," he says. "We also figured that *yacc* would work just as well with object-oriented code as straight C would." The switch was gradual; over time, Intek's original system evolved into a C++ system.

Although the lack of a C++ symbolic debugger for the 386 was at first a major inconvenience, Mac was able to set breakpoints on C++ source because the Intek translator uses the *#line* preprocessor. The biggest problem was the way C++ mangled all the system names. To get around the headaches this generates, they now dump a symbol table out with the map file. The C debugger sees the name as mangled, but using a symbol map still

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

solves the problem. Mac points out that AT&T also supplies a mangler/demangler piece of code.

Intek rarely needs debuggers now. "Our code mainly does the event loop," he says. "The DOS extenders provide some help for debugging through the protected mode, and they're nice enough to tell you what happened when things go awry."

Recently, Intek has been using Borland C++ for prototyping and for small projects. All production work is done with Intek C++ and either Watcom's or MetaWare's C-386. Much of the work uses Novell's B-trieve, which is complicated by a great deal of mode-switching between real and protected modes. Mac observes that the company's 32-bit extended DOS applications have proven to be stable. Turbo and Borland C++ have "raised the tide" by exposing people to the language, Mac believes. In his view, the market has been both improved and expanded by Borland's entry into the marketplace.

BUT WHAT ABOUT YOU?

Of course, not everyone is involved in devising systems to meet huge inventory-control requirements. Is Mac's experience unique to his market niche? Is the nature of his business particularly well-suited to the strong points of C++? Mac says that he agrees with the observation that Peter Drucker makes in his book, *Management:* "American industry defies generalization."

"No product can satisfy all needs," Mac says. "The markets we serve are highly individualized, and with C++ we can customize quickly." In real-time systems, a significant amount of "day-after stuff" is involved. A customer might change their mainframe system, for example, requiring the company to rework the interface to the database system.

Thorough testing is important in this market. The systems Intek writes are frequently in mission-critical situations, where lives—not just dollars—are on the line. Mac notes that this has forced his company to be "way up in the 99th percentile" in meeting specifications.

The company sometimes builds models of its target-project hardware. These prototypes, made of switches mounted on plywood and connected to I/O subsystems, are simulations of the manufacturers' automated equipment. Most of the Intek devices are controlled using non-standardized communications protocols. The company's device library contains more than 100 kinds of devices. Mac points out that the class name "device" fits over the top of all these things and can be used by event managers and system managers. Thus, adding a new device has little impact on the application code. Intek has licensed its suite of libraries to various companies, including Apple Computer and NovAtel Communications.

STREAMLINING BIG SYSTEMS

Intek has about one seven-figure software project a year; so far Balboa has been its largest project. Another big project was a

Tired of the Way CONFIG.SYS Boots You Around?

Then Give Your CONFIG.SYS the Boot--With BOOTCON.

BOOTCON

". . . it's bulletproof ... The interface is well thought out. You need this program."
Dr. Jerry Pournelle, Byte, July 1990

"Highly recommended."
John C. Dvorak, PC Magazine, July 1990

How many CONFIG.SYS and AUTOEXEC.BAT combinations do you have on your PC? BOOTCON eliminates the confusion by storing all your configuration information in just one set of configuration files. Each time you boot your PC you can select the configuration you want to use from a menu of up to 26 choices.

BOOTCON is ideal when you need to use conflicting programs such as Windows 3.0, QEMM/386, 386max, Soft-ICE, and Lotus 123. A default configuration will automatically be used if you do not make a selection within the specified time-out period. BOOTCON also provides an optional password protection feature.

BOOTCON lists for **$59.95.**

MODULAR SOFTWARE SYSTEMS
(818) 440-9104 FAX (818) 440 9240
115 W. California Blvd. Suite #113, Pasadena, CA 91105

BOOTCON comes on both 5 1/4" and 3 1/2" disks. It has an unconditional 30-day money-back guarantee. MS-DOS or PC-DOS 3.10 or later is required.

Windows and MS-DOS are trademarks of Microsoft Corporation. PC-DOS is a trademark of International Business Machines Corporation. QEMM/386 is a trademark of Quarterdeck Office Systems. 386 MAX is a trademark of Qualitas Corporation. Soft-ICE is a trademark of Nu-Mega Technologies. Lotus and 123 are trademarks of Lotus Development Corporation.

Circle **44** on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

RS-232^3

Three powerful tools for programming asynchronous communications
The Grunt Work done – by Greenleaf

NEW C Version!

Greenleaf CommLib Level 2™ Version 3.1

- ✓ New **Level 2** device independent function calls supported by six drivers – Greenleaf, Fast, Polled, Smart DigiBoard, MODEM Assist, FOSSIL. More on the way.
- ✓ XMODEM, YMODEM (Batch), ZMODEM (Batch), Kermit, ASCII file transfers. 1K, G, CRC options.
- ✓ Inteligent DigiCHANNEL™ support reduces loading on PC.
- ✓ Access modems via network using Fresh Technology MODEM Assist™
- ✓ XON/XOFF, RTS/CTS and DTR/DSR flow controls.
- ✓ Up to 115,200 baud. 8250, 16450 UART, and now16550 FIFO modes. Ring-bufferred, all four interrupts serviced for all ports. Buffers to 65K. New Fast receive interrupt option for tight spots.
- ✓ Unlimited number of ports for ISA and MicroChannel, COM1..COM8, all popular multi-port boards, including Intelligent DigiBoard.
- ✓ More than forty modem control functions.
- ✓ Includes keyboard functions, Ctrl-Break Handler, unique WideTrack™ Receive feature. Option to filter up to 3 codes from input.

NEW C++ Product!

Greenleaf Comm++™

- ✓ True C++ Class Library
- ✓ Device independence through inheritance.
- ✓ Supports Zortech C++ for DOS, OS/2.
- ✓ IBM PC, XT, AT, PS/2 and Compatibles
- ✓ COM1..COM4
- ✓ Baud rates to 115,200
- ✓ XON/XOFF, RTS/CTS Flow Controls
- ✓ XMODEM, Kermit File Transfer protocols
- ✓ VT52, VT100 subset, ANSI Terminal Emulation plus TTY emulation
- ✓ Hayes modem control classes
- ✓ Line and link level controls & status
- ✓ Extensive examples help you get started with OOP applications using communications.

Debugging Tool!

Greenleaf ViewComm™

- ✓ Data Line Monitor - Full featured RS232 "Datascope" *with graphic breakout box screen* runs on your PC or Laptop.
- ✓ Monitor, Capture Data in ASCII, EBCDIC, Baudot to 115,200 baud.
- ✓ Capture to buffer or file, Review, String Search
- ✓ Unlimited Triggers on Modem Status, Data Patterns, including wildcards
- ✓ View Data in Hex, Octal, Decimal, Binary, with Graphic Characters for Control Codes (VGA, EGA)
- ✓ Timestamps, Variable Resolution, for each character, down to 1 msec.
- ✓ Source Mode lets you send from file, keyboard or both
- ✓ File format & include files provided so you can analyze captured data.
- ✓ ViewComm will pay for itself the first time you use it.

All Greenleaf Libraries Come With:

- ✓ Free Source Code, Free Tech Support, Free BBS, Newsletter, Reference Guide
- ✓ Online Documentation System for MS Advisor, Norton, PDQPlus (With engine)
- ✓ Top Rated Documentation
- ✓ Support all major C and C++ compilers for DOS
- ✓ Exclusive **Gold Support Card** support available (Extra charge) guarantees all upgrades shipped free of additional cost, plus 800-numbers for Tech Support and 9600 baud BBS.
- ✓ No Royalties, Money-back guarantee, of course.

❝ *If you are seriously contemplating writing an application that uses interrupt driven asynchronous routines, you absolutely can't go wrong with CommLib from Greenleaf Software."* - Telecomputing Magazine

❝ *Greenleaf CommLib is the smoothest toolkit I own. 'OUTSTANDING' sums it up!"* - Larry Dalton

❝ *When developing communications software, there often comes a time when your software just doesn't quite work and it can be devilish to divine just what the problem is. Greenleaf has a simple, inexpensive, and remarkably effective solution. The product is ViewComm."* - Boardwatch Magazine

☎ Call today for complete information, demo or to order. MasterCard, VISA, AmEx, approved Purchase Orders.

Greenleaf CommLib v3.1	**$359**
Greenleaf Comm++ v1.0	**$199**
Greenleaf ViewComm v3.00	**$395**

1-800-523-9830

214-248-2561 FAX 214-248-7830

BBS (Free Demos) 214-250-3778

Greenleaf Software Inc.

16479 Dallas Parkway, Suite 570

Dallas, TX 75248

GREENLEAF Software ©

PRODUCED BY APPLE.NZ.ORG

ELECTRONIC REPRODUCTION PROHIBITED

Circle 45 on Reader Service Card

TAKE THE LANGUAGE TO THE LIMIT.

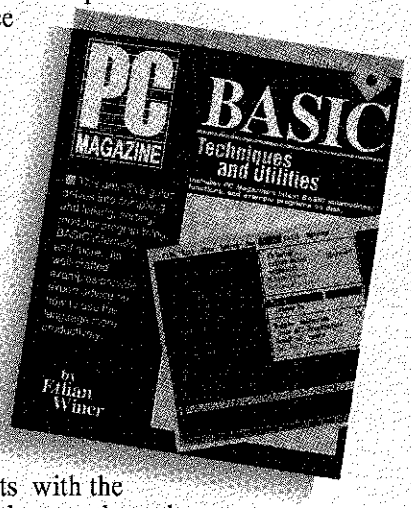
Two New Information-Packed Sourcebooks with
On-Disk Tools from *PC Magazine*.

PC MAGAZINE BASIC
Techniques and Utilities
• *by Ethan Winer*

PC Magazine BASIC Techniques and Utilities gives you the definitive guide to programming in BASIC, written by *PC Magazine*'s contributing editor Ethan Winer. Winer provides useful real-world advice as well as tips to increase programming productivity.

On-Disk, Ready-to-Use Utilities and Examples

In addition to Winer's information-packed guide, you get a 5 1/4" disk full of useful subroutines, functions, and examples. Included are DOS and BIOS service utilities, input and menu routines, windowing and video routines, and a complete terminal program with phone dialer.



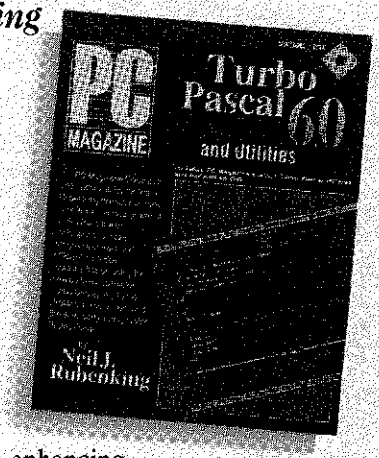
Ethan Winer has written the most useful book available on BASIC, providing helpful insights on BASIC 7.0, 7.1, and QuickBASIC. He starts with the essentials and carries the user through modular programming, compiling and linking, memory management, accessing DOS and BIOS services, and a host of other informative topics.

only $29.95

The Book on BASIC

A well-respected authority on compiled BASIC, Winer is the author of the QuickPak Professional and PDQ software packages. As a contributing editor to *PC Magazine*, he writes regularly on programming-related subjects.

PC MAGAZINE Turbo Pascal 6.0
Techniques and Utilities
• *by Neil J. Rubenking*



Here's a sourcebook and disk that will move the experienced programmer to expert and the expert to master. Written by *PC Magazine*'s technical editor, Neil J. Rubenking, the book offers inside tips, advice, and numerous units, objects, and routines.

Rubenking takes you through object-oriented programming, productivity-enhancing techniques, and power routines for important programming areas such as I/O, program flow control, and string handling.

only $39.95

Hands-On Examples and Utilities Disk

PC Magazine Turbo Pascal 6.0 Techniques and Utilities comes with a 5 1/4" disk loaded with power source code utilities and examples. It helps you get further into the language quicker.

Real-World Solutions to Real-World Challenges

As the writer of *PC Magazine*'s popular "User to User" column, Rubenking has answered hundreds of questions for readers. He knows the challenges that Turbo Pascal programmers face, and he's written the sourcebook to help you meet them.

Call 800/688-0448

For more information or for the location of the ZD Press bookseller nearest you

ZIFF-DAVIS ZD PRESS

Circle 50 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

system to handle the parts used to build Macs at Apple's Building 5 in Fremont, California. Most parts are received and placed in motorized carousel bins by Cartesian robots. Parts are ordered on the manufacturing line and delivered from the carousel system to the floor via robotic automatic guided vehicles (AGVs). After the receiving process, the inventory isn't touched by human hands until it gets to the factory floor.

Apple wanted the job done using Macs. "Not many people say they want their factory done on Macs. Everyone else who bid on the job must have insisted on using a VAX, so we were the ones who got it," Mac says. All the networking was done on Appletalk, which was difficult because of the number of nodes required and the speed that was needed. Mac notes, "The level of detail necessary for this inventory would be impossible for people to keypunch in."

This system has greatly increased efficiency. Before the Intek design was implemented, the Building 5 system had to keep roughly $8 million of inventory on hand. Now, $4 million is adequate. And inventory discrepancy amounts to only about $30 over a year's time.

Intek also designed an inventory system for Pep Boys, an automotive parts retailer with a distribution center in Mesquite, Texas. The Pep Boys system is based on zones of automatic storage and retrieval systems (ASRs). Again, Mac's group mapped out a system that uses carousels resembling the moveable clothing racks used by dry cleaners. Steel baskets are suspended on a track 6 feet high and 70 feet long; each basket can handle as much as 1,000 pounds of material. The inventory worker stands in one place and is prompted by lights and displays to pick items from the carousels and place them into bins for distribution. The industry average for order picking is about 150 line items an hour. With the Intek system, however, the Pep Boys workers average an "astounding" 900 to 1,000 line items per hour.

TAKING CARE OF BUSINESS

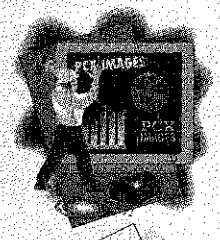
Mac enjoys seeing Intek's C++ tool being adapted by other groups to fit their specific requirements. For example, one Intek customer is using the Intek C++ tool for cross-compiling onto the TI 34020 graphics coprocessor.

Being in the software development market has the additional advantage of keeping Mac up-to-date on what other developers are working on. For the future, Mac hopes to build a CASE-like development environment that would allow the invention of new classes and the extraction of existing ones.

"Our work in the C++ business has helped our other business considerably," he says. Although he admits to knowing more about the internals of the AT&T parser than he'd like to, he sees more advantages to C++ than disadvantages.

In addition to its allowing the reuse of existing C code, Mac cites C++'s ability to intermix with C and the opportunity it gives programmers to ease into OOP. "C++ really does provide reusability with reality," he says. "Some purists complain that it 'shoehorns' object-orientedness into C. But the truth is, you can gradually evolve from C to C++ without betting your enterprise on it." DJ

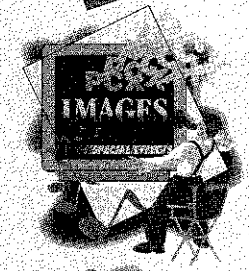
ADD THE POWER OF GRAPHICS TO YOUR PROGRAMS WITH GENUS



PCX PROGRAMMER'S TOOLKIT 4.0

Finally, a programmer's toolkit that helps you incorporate graphics into your programs quickly and easily. The PCX Programmer's Toolkit includes over 90 routines to display, save, scale, and print PCX bitmapped graphics from almost any program.

ONLY $249



GX EFFECTS 2.0* NEW!

Add spectacular special effects (F/X) to your programs, without being trapped in someone else's editor or slideshow program. Wipe, split, crush, slide, sand, drip, diagonal, spiral, random, or explode your graphics. Includes new animation support and sprite capabilities.

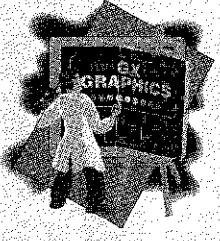
ONLY $199

*PCX Toolkit, GX Graphics, or GX Text recommended but not required.

GX TEXT 2.0 NEW!

Now you can display blazing bitmapped text in any graphics mode as simply as text mode. With GX Text, you can display your textual information right along with your graphical data, using the display or virtual buffers. Includes a graphical font editor and 1MB+ of fonts.

ONLY $149



GX GRAPHICS 1.0

A complete graphics library supporting all graphics primitives. Use the GX Graphics toolkit instead of the BGI or MS libraries and make your program faster, smaller, and more portable across compilers – while supporting more video modes.

ONLY $199

TO ORDER ANY OR ALL OF THE GENUS GX DEVELOPMENT SERIES, CALL TOLL-FREE:

1-800-227-0918

VISA, MASTERCARD, AMERICAN EXPRESS, CHECK, C.O.D., PURCHASE ORDER

G·E·N·U·S MICROPROGRAMMING

2900 WILCREST • SUITE 145 • HOUSTON, TX 77042 • (713) 870-0737
FAX (713) 870-0288 • BBS (713) 266-9362

Circle 47 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The word is OOP

By M. Steven Baker

One of today's most popular computer buzzwords is object-oriented programming (OOP). We now have object-oriented design, analysis, languages, tools, environments, and databases, to name a few. The high level of interest in these topics is being fed by and parallels the explosion of books about them. This time around I've chosen to review a modest handful of the new titles in this arena. I'll mention as well some new computer language books that may also be of interest.

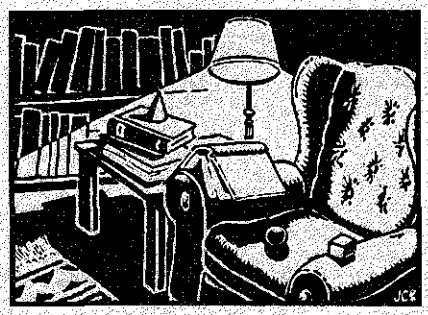


ILLUSTRATION BY JIM CARPENTER

OOP WITHOUT SERMONS

Programmers at the beginning of the OOP learning curve face a formidable process and can benefit from some even-keeled discussions of OOP concepts that steer clear of the proselytizing found in many books. An excellent starting point is ***Object-Oriented Design with Applications,*** by Grady Booch (Benjamin Cummings, 1991). The author provides readers with a sound understanding of fundamental concepts of the object model and outlines the process of object-oriented design (OOD) without attaching these ideas to a single language.

Although Booch's name is most often associated with Ada, this book is targeted for OOP programmers of all flavors. Readers will appreciate the author's clear understanding of the object model and of the most common object-oriented languages: Smalltalk, Object Pascal, C++, Lisp, and Ada.

Along with Booch's sleek prose, you'll find humorous illustrations, valuable figures, and useful notes. I'd love to duplicate one of the humorous drawings but will have to make do with the epigraph for the first chapter:

A physician, a civil engineer, and a computer scientist were arguing about what was the oldest profession in the world. The physician remarked, "Well, in the Bible, it says that God created Eve from a rib taken out of Adam. This clearly required surgery, and so I can rightly claim that mine is the oldest profession in the world." The civil engineer interrupted and said, "But even earlier in the book of Genesis, it states that God created the order of the heavens and the earth from out of chaos. This was the first and certainly the most spectacular application of civil engineering. Therefore, fair doctor, you are wrong: mine is the oldest profession in the world." The computer scientist leaned back in her chair, smiled, and then said confidently, "Ah, but who do you think created the chaos?"

The book is well organized into three major sections: Concepts, Methods, and Applications. Each chapter is summarized and concludes with a description of further readings. The concepts section has chapters that discuss complexity, the object model, classes and objects, and classification. The focus is on using the object model to deal with "industrial-strength" software—large, complex problems.

The next section discusses methods of object notation, the design process, and pragmatics. Booch proposes a method for the development of complex systems using a graphical notation for OOD, followed by the design process itself. A thoughtful section on the benefits and risks of object-oriented design is also included.

The last part of the book looks at applications. The author presents OOD at work, with realistic problems using Smalltalk, Object Pascal, C++, the Common Lisp Object System (CLOS), and Ada. A chapter outlines the OOD process by using Smalltalk to implement a home heating system.

The chapter on Object Pascal employs MacApp and the Macintosh user interface to illustrate a scientific application—creating a geometrical optics construction kit for analyzing lenses (ray tracing). C++ and SQL are used to implement a problem-reporting system (database). A cryptoanalysis system to solve simple ciphers is implemented using CLOS. Ada is used to develop a traffic-management system. Each of these chapters features clear illustration of both the OOP language and the environment.

A valuable appendix contains information on object-based and object-oriented programming languages. A genealogy is presented for the five languages used in the large applications. These same five languages are then compared feature-for-feature and are used to implement the same shape-drawing class. For both its content and its style, I can't recommend this book highly enough.

OPENING UP OOP

Another good starting point for OOP study is Addison-Wesley's ***An Introduction to Object-Oriented Programming,*** by Timothy Budd (1991). The author's earlier book, ***A Little Smalltalk*** (Addison-Wesley, 1987), was noteworthy for its explanation of OOP concepts by the construction of a simple Smalltalk interpreter. By chance I discovered that Budd teaches at Oregon State University, in Corvallis—less than an hour downstream from *PJ*'s office on the Willamette River.

As was the case with his earlier book, the inspiration for this new title arose when the author was faced with teaching an OOP course and could not find a suitable text that wasn't language-specific. Although some competent OOP books have been published, most are oriented toward a single language and give the impression that OOP is synonymous with "object-oriented programming in *X*," where *X* happens to be the author's favorite language. This book, however, introduces OOP concepts without focusing on a single language implementation. Parallel examples of the major OOP

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

concepts are presented in C++, Smalltalk, Objective-C, and Object Pascal.

The first two chapters focus on object-oriented thinking independent of language syntax. The author presents a useful scheme to develop classes using index cards. Subsequent chapters focus more on the mechanics of OOP to illustrate methods, message passing, instances, initialization, inheritance, binding, polymorphism, and encapsulation and data abstraction using classes. The reader is brought up to speed gently before the plunge is taken into language syntax. Interspersed throughout are case studies that present small but complete application examples to illustrate important points.

More academic than those in the Booch book, the case studies consist of the "Eight Queens" puzzle, mixed-type arithmetic, solitaire (C++), a billiards game (Object Pascal), container classes (C++), and collections (Smalltalk). Each major OOP concept and the Eight Queens case study are illustrated in C++, Smalltalk, Objective-C, and Object Pascal. By providing four different solutions to the same problem, the book gives readers a chance to compare and contrast various features of these OOP languages.

A short chapter on implementation discusses how compilers handle various OOP concepts, including virtual method tables, dispatch tables, and name encoding. Appendices contain complete source code for the case studies. I recommend this book as a thorough, readable introduction to OOP.

YOUR LANGUAGE OR MINE?

Of course, your local bookstore carries many new titles on C++ and other OOP languages for the hungry initiate. C++ programmers have trouble keeping up with changes in the language. Some of last year's better C++ titles cover the language as defined by AT&T's CFRONT translator Version 1.2. Other books focus on a particular PC implementation such as Turbo C++ or Borland C++. From my perspective, these spend too much time on the underlying C language routines supported by the particular development environment, and too little time on the OOP features.

Addison-Wesley has published a number of new and revised titles written by the folks at AT&T Bell Labs. Bjarne Stroustrup, the primary author of the C++ language, has a revised edition of ***The C++ Programming Language.*** This book is considered on a par with Kernighan's and Ritchie's classic ***The C Programming Language.*** I found both the original and this revised Stroustrup book dry and somewhat formidable for the OOP novice. This edition—about twice the size of the 1986 edition—covers many of the new features that have been added to the language in the past five years. One drawback to using the second edition is that a number of features Stroustrup uses heavily are considered experimental. You won't find them in any real-world C++ compiler—some aren't even in AT&T's soon-to-be-released C++ v3.0 translator. To compound this problem, the text doesn't designate these features (templates and error handling, for example) as experimental.

The Annotated C++ Reference Manual, by Margaret Ellis and Bjarne Stroustrup (1990), is the definitive reference to the language through February 1990. This book is even drier (skip the silica gel) and is useful only to those who already know the language. But for language implementers, this is the definitive text. Templates and exception handling are described in this annotated reference manual—even though you won't find a DOS compiler that supports them yet.

GETTING TO KNOW C++

Addison-Wesley's ***C++ Primer,*** by Stanley Lippman (also from AT&T Bell Labs), provides a somewhat more digestible introduction to the language. The author covers C++ through the modifications and features available on AT&T C++ Release 2.0 (multiple inheritance and typesafe linkage, for example). True to the Unix tradition, this book begins with Chapter 0. The first three chapters cover the essential elements of a C++ program (much of this material reviews the C language) and touch briefly on various OOP features included with C++. The remaining five chapters focus more on the object-oriented features of C++, including overloading, classes, member functions, class derivation, and virtual functions. Appendices discuss the C++ I/O library, future directions for the language, compatibility of C++ with C, and compatibility with C++ Release 1.2.

Wiley's ***Mastering C++,*** by Cay Horstmann, provides a thoroughly readable introduction to C++, with an emphasis on techniques used in real-world programs. Horstmann does not rely on the contrived examples used in many books and doesn't assume that the reader is a C programmer. "Pascal Notes," sprinkled through the text, aid Pascal and Modula-2 programmers in the transition to C/C++ syntax.

Mastering C++ has a straightforward organization for the C++ initiate. After a quick tour through the language, the author covers data, functions, pointers, arrays, references, and structured types. The next six chapters focus on the OOP features, including overloading, I/O, memory management, inheritance, modules, and generic classes. Examples of object-oriented design—including a simulation, a home accounting system, and a matrix calculator—round out the book. Most chapters conclude with a useful section on potential pitfalls. This book is easier to comprehend than the drier tomes from AT&T.

OOP MEETS THE MAC

Microsoft Press's ***Object-Oriented Programming Power,*** by Chuck Sphar, is useful for programmers who use THINK Pascal and Object Pascal on the Apple Macintosh. The author provides a thorough introduction to OOP, along with concrete examples of how to use application frameworks (such as the THINK C class library and MacApp) and how to write reusable software components. Continuing in the lighthearted tradition, this book's figures and examples are humorous but useful. The introductory chapter opens with the following code snippet:

```
if (you.ReadyToLearn('object-oriented
    programming')) then
if (you.LanguageOf = Pascal) and
      (you.MachineOf = Macintosh) then
you.Need('this book');
```

The book is divided into three parts: OOP concepts, building object-oriented applications on the Macintosh, and working with polymorphic software components. A tiny application framework, PicoApp, is used for illustration and to develop two other, more complete

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

ADVERTISER INDEX

Reader Service	Advertiser Name	Page Number
17	American Cybernetics	25
43	Audio Solution, The	68
21	Austin Code Works, The	33
41	Autumn Hill Software	66
110	Az-Tech Software, Inc.	86
24	BCM	39
13	Black Ice Software	22
3	Blaise Computing, Inc.	7
56	Borland International	C4
19	Block Island Technologies	32
115	Burton Systems Software	86
*	C.C. Software	86
37	Catenary Systems	62
*	C Gazette	83
*	Copia International LTD.	84
12	Data Techniques, Inc.	22
123	DatTel	87
114	Discus Distribution Service	86
127	Extra Mile Writing Services	87
112	EZX	86
28	General Software	50
47	Genus Microprogramming, Inc.	73
*	Gimpel Software	35
45	Greenleaf Software, Inc.	71
35	GUI Computer	61
33	Helpful Programs, Inc.	57
118	Impulse Engineering	86
113	Interstitial Software Systems	86
14	Jensen & Partners International	23
31	KADAK Products, LTD	54
11	Kandu, Inc.	21
5	Knowledge Dynamics Corp.	8
25	Lahey Computer Systems, Inc.	39
120	Madden Associates	86
55	Meridian Software Systems	C3
8	MetaWare Incorportated	11
*	Microsoft Corporation	5
*	Microsoft Corporation	15
128	Microtools	87
27	Mix Software	46
44	Modular Software Systems	70
38	Nanosoft	62
2	Nu-Mega Technologies	2
124	Olivadoti Associates, Inc.	87
111	Opt-Tech	86
15	Opus Software	24
20	Pathfinder Associates	32
39	PC Tech, Inc.	63
46	PC Techniques	78
9	Periscope Company, The	14
*	PJ Mail Order Store	77
6	Programmer's Corner, The	8
40	ProGraphx Toolbox	63
125	Quaid Software Limited	87
29	Quantasm Corporation	51
1	Quarterdeck Office Systems	C2, 1
*	Que Corporation	85
30	Quibus Enterprises	52
*	Roundhill Computer Systems	84
32	RSS Engineering	55
117	Ryle Design	86
23	Seabreeze Software Systems	37
54	Sequiter Software, Inc.	65
*	Set Laboratories, Inc.	21
7	SLR Systems	10
49	Software Blacksmith	79
51	Software Development '92	81
119	Software Mart	86
34	Software Success	57
121	Software Toolz, Inc.	87
36	Softway, Inc.	61
26	Stepstone	43
42	Strategic Software Designs Inc.	68
10	Sub Systems	19
126	Sutrasoft	87
16	TEGL Systems Corp.	24
18	TurboPower Software	28
122	Upper Deck Systems	87
4	Utilis, Inc.	8
*	V Communications	47
116	Wenham Software Co.	86
22	Western Wares	36
48	Willies' Computer Software	79
*	**Windows Tech Journal**	29
50	Ziff-Davis Press	72

When contacting any of the above advertisers, please tell them you saw their ad in ***Programmer's Journal*** and that you appreciate their continuing support of this magazine. The advertisers marked with an asterisk (*) prefer to be contacted directly. ***PJ***'s Advertiser Index is provided as a free service, and ***Programmer's Journal*** cannot be held responsible for omissions or errors. ***Programmer's Journal*** is not responsible for errors in advertisements.

EDITORIAL INDEX

Reader Service	Vendor Name	Page Number
221	Borland International	82
220	Decision Software	82
219	Genus Microprogramming	82
218	Greenleaf Software	80
222	Media Cybernetics	82
214	Meridian Software	80
226	MetaWare Inc.	84
225	Microsoft Corp.	84
213	OPUS Software	47
215	Periscope	80
228	Phar Lap Software	84
216	Quantasm	80
224	Rational Systems	82
212	SLR Systems	47
223	Sycamore Microsystems	82
217	Watcom	80
227	Whitewater Group, The	84

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

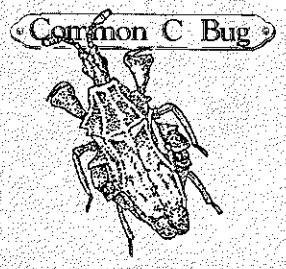
PJ ISSUE SPECIAL!

SAVE 13%

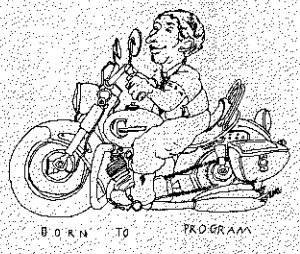
Multi-color imprints on the highest quality 100% cotton T-shirts.

SIZES: S, M, L, XL, XXL

$14.95 each — includes U.S. shipping



COMMON C BUG (beige)



BORN TO PROGRAM (black)

ALL T-SHIRTS NOW ONLY $12.95 EACH!

PJ T-shirts make great holiday presents for your friends—or for yourself. Order as many as you want, but order today to avoid the rush.

Free U.S. shipping. Add $2.50 shipping per shirt to Canada or Mexico, $5.00 shipping per shirt to any other foreign destination. Offer expires 12-31-91.

PARALLEL PORTS (wine)

IT'S A FEATURE
(yellow, hot pink, blue, black)

← **BUGS RULE**
(white with 4 color art)

$7.00 each PREPAID ONLY

A few of the Back Issues currently available

Vol. 8.5: OBJECT_ORIENTED PROGRAMMING: More colors on the VGA; A virtual memory system in C/C++; Pitfalls in Turbo Pascal OOP; Writing applications that travel; Late binding; Tools for 386 DOS; Windowing text with X; Cache memories; Breaking the rules again; SPARCs plot world takeover.

Vol. 8.6: THE DATABASE CHALLENGE: VGA color paging; The database dilemma; Networking databases; B-tree indexing; Virtual memory paging; Writing an ANSI C preprocessor; Peak performance: On to the 80486; Calendar conversions; Software components; Optimization strategies; Pink noise; Oh say can you C?

Vol. 9.1: GRAPHICS AND WINDOWING: VGA color cycling; Digital typography; Windowing for two; GUI madness; Text buffers in windows; What's on the menu; 386 DOS meets ANSI C; Optimania; Real-mode virtual memory; Enhanced keyboards; An open look at Motif; Negotiating tricks and traps; For a few good books, Goliath beats David—As usual.

Vol. 9.2: LANGUAGES: Windows 3 input hooks; C compilers for Windows; Fortran 90 is coming!; Pragma interface; String handling; Porting to the Mac; Disassembler roundup; Exploring XMS; Using EMS for fast file I/O; Around the bend again; Pink noise; Language lore; Effective marketing presentations; COBOL revisited; Language parsers and Fortran tools; Changes in the fast lane.

Vol. 9.3: HARDWARE FOR DEVELOPERS: Pixel panning; Hardware-assisted debuggers; Object-oriented SCSI programming; Numeric exception handling; Virtual realities; BASIC libraries; BIOS compression; Hashing functions; Computer stores and popcorn kernels; Programmer productivity; Language translation and 386 debugging; Hardware help; Hardware decisions.

Vol. 9.4: NETWORKING AND COMMUNICATIONS: A coat of many colors; Communicating over NetBIOS; Spying on NetBIOS; Window viewports; Loop unrolling; A RISCy business; Data structure imperatives; Text filters in BASIC; Pink noise; Don't be wrong about your rights; Network madness; Network debugging and comm libraries; Benchmark SPECulations.

Vol. 9.5: OPERATING SYSTEMS: Eavesdropping on I/O; DOS extenders; Managing memory; DESQview-aware programs; Windows property lists; Icon power; Divide and conquer; Window layers; TQM: The new Beltway buzzword; Chips off the old block; Pursuing systems programming; Thinking about project management; A software fable; Playing for keeps.

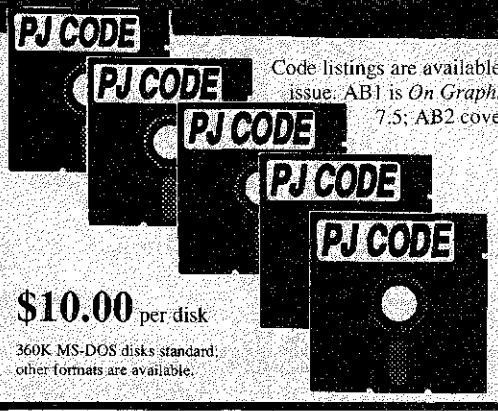
BACK ISSUE SPECIAL

A complete set of back issues! Only $99.99 plus shipping.

10% discount on any order of 10 or more T-Shirts, Disks or Back Issues.

All mail orders must be accompanied by check or Money Order -- PREPAID ONLY.

Use the order form on the card opposite.
Order by phone with your VISA/MC.
CALL TOLL FREE
800/234-0386 ext 53



$10.00 per disk

360K MS-DOS disks standard; other formats are available.

Code listings are available from volume 5.5 to the current issue. AB1 is *On Graphics* series code from issue 5.1 to 7.5; AB2 covers issue 7.6 to the current issue.

Diskettes are available for the following code:

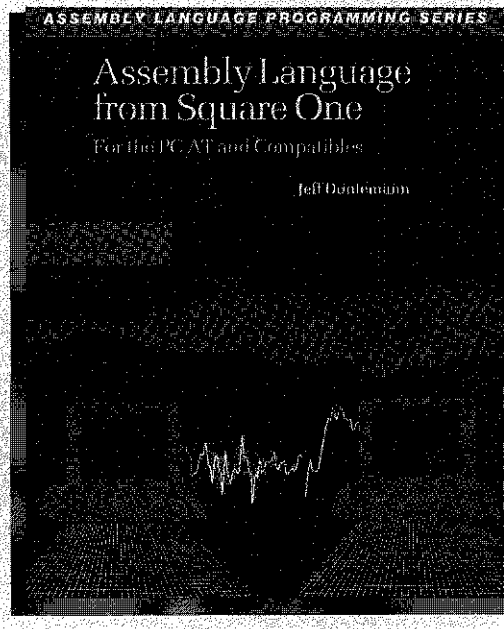
AB1	6.4	7.5	8.6
AB2	6.5	7.6	9.1
5.5	6.6	8.1	9.2
5.6	7.1	8.2	9.3
6.1	7.2	8.3	9.4
6.2	7.3	8.4	9.5
6.3	7.4	8.5	9.6

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

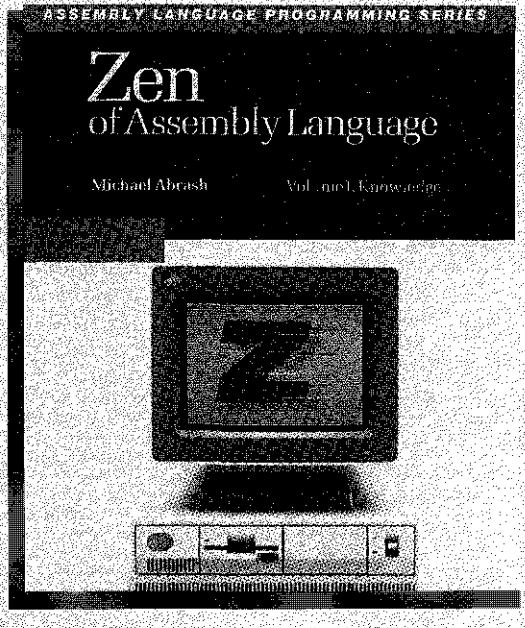
The Best Books from PC TECH

Take It To The Limit!

Jeff Duntemann taught you object-oriented programming in the pages of *PC TECHNIQUES*, and he can teach you assembly as your first programming language. Jeff's crisp metaphors and offhand, humorous style carries even the very toughest concepts home. After reading this book you'll be writing programs in assembly, but more important, you'll know both what you're doing and why you're doing it. **Every copy signed by the author!**



Order *Assembly Language from Square One* from PC TECHNIQUES and receive the listings diskette for FREE!



"Pushing the Envelope" columnist Michael Abrash has made hand-optimized assembly code his trademark, and in this monumental volume shares the accumulated wisdom of his entire career as a performance programmer. Understand how the 80x86 family does what it does. Learn of the "cycle eaters" that slow your code down. Learn how to do what no higher-level language compiler can do. That's the Zen of assembly language. **Every serious programmer must have this book!**

Send Coupon or Call Today!—602-483-0193

Assembly Language from Square One $29.95 (Free Disk)

Zen of Assembly Language $29.95

Subtotal $_____
Shipping:
$2.50 per book in US.
$5.00 per book outside US.
$_____
Total $_____

Prepayment required—U.S. Banks only

___Check/Money Order ___VISA/Mastercard

Card #_____Exp. Date_____
Name_____
Company_____
Address_____
City/State/Zip_____
Country_____
Phone_____

PC TECHNIQUES, 7721 E Gray Rd., Suite 204, Scottsdale, AZ 85260

STOP!

Don't write another line of assembly code until you've read these books!

Circle 46 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

application examples (CrapGame and PicoSketch). The accompanying disk contains full source code for all the examples, including the PicoApp class library. Appendices cover porting code to other Macintosh Pascal compilers and include a bibliography and a glossary. The author provides a thoughtful touch in the annotated bibliography by describing books and articles that he found most useful.

NON-OOP LANGUAGES

Hot off the press, Prentice-Hall's ***The Standard C Library,*** by P.J. Plauger, is a fine addition to your bookshelf. The book shows how to use and implement the ANSI standard C library by providing readers with 9,000 lines of portable C code that implement the entire ANSI C library. You'll also find a thorough discussion of newly added features, including multiple locales and large character sets (such as Kanji). The code has been tested with C compilers from Borland, Saber, Free Software Foundation (Gnu C), Sun, Unix, and Vax Ultrix, and it passed the Plum Hall suite of validation tests. A companion disk (available at additional cost) contains all the source code from the book. *The Standard C Library* is an excellent reference to the ANSI C library and a superior example of portable C programming. It deserves a place beside Harbison and Steele in every C programmer's book collection.

For those who prefer the other PC language, Ziff-Davis Press has a winner in ***BASIC Techniques and Utilities,*** by Ethan Winer. This book is a detailed, comprehensive tutorial describing the inner workings of Microsoft's BASIC compiler. The first section ("Under the Hood") explains how Microsoft's compiler translates BASIC source code into 80*x*86 assembly language and how compiled code interfaces with the BASIC runtime libraries. It also contains a useful and exhaustive discussion of data structures, variables, and data storage in BASIC that has never been documented by Microsoft. Using actual disassemblies of compiled code, Winer compares various flow-control methods for code size and execution speed.

The second section ("Programming Hands On") focuses on practical aspects of BASIC programming. One handy chapter deals with debugging using both the QUICK environment and CodeView. The final section, titled "Beyond BASIC," has chapters on material rarely covered in BASIC language texts. These include ROM BIOS data areas, DOS and BIOS services available using CALL INTERRUPT, and a tutorial on 80*x*86 assembly language from the BASIC programmer's perspective. The accompanying disk includes prebuilt libraries, all the example programs, and the assembly language routines. This highly recommended book finally provides something for advanced BASIC programmers.

WCSC Professional Development Tools

COMM-DRV
- Professional serial communications library.
- Device Driver/Int21,Int14,FOSSIL
- De-installable TSR w/INT14,FOSSIL
- 25+ Ports, irq Sharing, remap ports
- Hdw./Softw. Handshaking/115 kbaud
- 8250/16450/16550(Auto Detection)
- Desqview,Windows 3.0,pcAnywhere

Device/TSR/Libraries/Source $89.95

COMM-LOG
- Background file transfer TSR
- Xmodem(1k)/Ymodem(Batch/G)
- Background logging to disk TSR
- Application & Command Line Interface
- File transfer/logging on seperate ports / seperate files concurrently. Easy to use.
- TSRs are de-installable
- Requires COMM-DRV

TSR & File Transfer Sources $89.95

SAVE!!! Buy 2 products for $159.95, 3 for $225.95, 4 for $275.95

MTASK
- Preemptive Multitasking TSR)
- Allow creation of several threads of execution in your application.
- Threads may make DOS calls.
- Threads may sleep or be awakened
- Threads may have different time slice
- Intertask message support

TSR and Examples $89.95

SCRN-INPUT
- Professional screen & keyboard library
- Create screens on monitor or over serial ports
- Save/Restore/Scroll screen areas
- Complete cursor & keyboard control
- Data driven input field screens
- Absolutely no royalties
- Extreme ease of use/Many examples

Library and Sources $89.95

4 Port Serial Card w/16450 $110 w/16550 $170

WCSC

1-800-966-4832

Willies' Computer Software Co.
2470 S. Dairy Ashford, Suite 188
Houston, Texas 77077

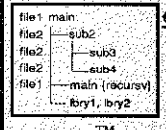
TEL (713) 498-4832
FAX (713) 568-3334
BBS (713) 568-6401

VISA/MC/AMEX/PO/COD/CHECK 24 Hours/Day 7 Days/Week

Circle 48 on Reader Service Card

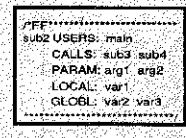
C AND C++ DOCUMENTATION

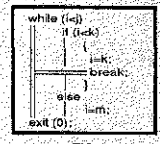
NEW! C-METRIC™ ($49) COMPLEXITY / QUALITY
- Calculates "Cyclomatic" path complexity for functions and system
- Counts Lines, Comments, Code, and 'C' Statements



C-CALL™ ($59) FUNCTION HIERARCHY
- Tree-Diagram showing function hierarchy
- Table-Of-Contents of functions versus files
- Detailed cross-reference of functions

C-CMT™ ($59) FUNCTION COMMENT
- Generate/Insert function comment blocks
- Re-run anytime to update comment blocks
- Retains any user-generated comments





C-LIST™ ($49) LIST OR REFORMATS
- Action-Diagrams show logic/control flow
- Reformats to various standardized formats
- Line numbers, page numbers, and titles

C-REF™ ($49) CROSS-REFERENCES IDENTIFIERS
- Summary list of Local/Parameter/Global/Define Identifiers used
- Detailed Cross-Reference of Identifiers vs Function/Line usage
- Produces Class-Hierarchy Tree-Diagram for C++ Classes

SPECIAL OFFER: ($189) Complete C-DOC™ Package
- All 5 programs fully integrated as 1 overall C-DOC program (DOS)
- NEW! special offer includes OS/2 protected-mode program
- Unconditional 30-DAY Money-Back Guarantee of Satisfaction !!

ACCURATE, and Fully AUTOMATIC (no source changes needed)

SOFTWARE BLACKSMITHS INC
6064 St Ives Way, Mississauga
ONT Canada L5N-4M1

VISA MasterCard
(416) 858-4466
Call for Immediate Shipment

Circle 49 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PRESSROOM

ADA 386 AND OPENSELECT CASE

Meridian Software Systems has introduced a pair of tools for PC Ada programmers. The products—Meridian Ada 386 ($1,695) and OpenSelect CASE ($795)—extend Meridian's PC-based Ada product line to support extended-DOS memory models and multiple-programmer projects.

Meridian Ada 386 uses Ergo Computing's OS/386 DOS extender to support global data, stacks, and objects greater than 64K. The compiler also generates 16-bit code. The package includes a source-code formatter and a cross-reference generator, as well as Adaorder, a **make**-like source-code analyzer that automatically compiles Ada modules in the correct order.

OpenSelect CASE is a multiuser, computer-aided software engineering tool for users of Ada and other languages. The analysis and design tool supports the Yourdon, Ward-Mellor, Hatley, HOOD, and Gane-Sarson methodologies. Additionally, OpenSelect CASE conforms to the CASE Data Interchange Format (CDIF), which means it can share project files with CASE tools from companies such as Cadre Technologies and Integrated Development Environments.

Meridian Software Systems, 10 Pasteur Street, Irvine, CA 92718, (800) 221-2522 or (714) 727-0700.

Circle **214** on Reader Service Card

PERISCOPE/REMOTE FOR PM

Periscope has entered the world of protected-mode debugging support with its introduction of three new hardware-aided debuggers. Periscope/Remote for OS/2 ($195), Periscope/Remote for Windows ($195), and the Periscope 32-Bit Toolkit ($495) allow programmers to debug protected-mode device drivers and operating systems. All three tools require a target computer connected to a host computer via null-modem cable, with a serial port on each system.

Periscope/Remote for OS/2 provides full-screen, source-level support for debugging OS/2 and PM device drivers under OS/2 1.2 and 1.3. It operates at the system (ring 0) level and can run concurrently with an applications-level (ring 3) software debugger. Periscope/Remote for Windows allows Windows programmers to debug device drivers and applications, including apps that generate Unrecoverable Application Errors. The debugger requires Periscope/32 software on the host and Windows 3.*x* (running in enhanced mode) on the target.

Using the Periscope 32-Bit Toolkit, developers of 32-bit operating systems, memory managers, and DOS extenders can debug their code. The Toolkit includes Periscope/32.

The Periscope Co., 1197 Peachtree Street, Plaza Level, Atlanta, GA 30361, (800) 772-7006 or (404) 875-8080.

Circle **215** on Reader Service Card

POWER LIB 2.0

More than 600 routines are packed into Quantasm Power Lib 2.0 ($99.95), **Quantasm**'s library for assembly language programmers. QPL 2.0 features a forms package, menu and windowing systems, more than 100 string-handling functions, extended-precision math functions, date and time functions, command-line parsing, sound control, and data compression. The library supports any size text screen as well as mice, DOS extenders, memory managers, and TSRs. It allows access from higher-level languages such as C, BASIC, Pascal, and Fortran. More than 6,000 lines of example code illustrate how the library routines are accessed. Full source is available for $299.95.

Quantasm Corporation, 19855 Stevens Creek Boulevard, Cupertino, CA 95014, (408) 244-6826.

Circle **216** on Reader Service Card

WATCOM C8.5, FORTRAN 77/386

Two new 32-bit compilers are available from **Watcom.** Watcom C8.5/386 ($995) is a complete implementation of the ANSI standard and includes a debugger, profiler, linker, **make,** and other tools. The bundled GUI and DLL kit allows programmers to create 32-bit applications for the Microsoft Windows 3.*x* environment. The system supports development under DOS, extended DOS, OS/2, and Windows environments, with target support for extended DOS (Rational, Phar Lap, and Ergo), Windows GUI applications, Windows DLLs, AutoCAD ADS, and embedded systems.

Fortran 77/386 (also $995) shares optimization technology and support tools with C8.5/386 and complies with the ANSI Fortran 77 standard (with Microsoft and VAX extensions). The compiler is runtime-compatible with C8.5/386, simplifying interlanguage function calls and mixed-language debugging.

Each compiler includes a royalty-free 32-bit DOS extender, DOS/4GW, from Rational Systems. DOS/4GW supports the DPMI, VCPI, and XMS standards.

Watcom, 415 Phillip Street, Waterloo, Ontario, Canada N2L 3X2, (800) 265-4555 or (519) 886-3700.

Circle **217** on Reader Service Card

GREENLEAF COMM++

A C++ class library for data communications has been released by **Greenleaf Software.** Greenleaf Comm++ ($199, including source) handles asynchronous communications via RS-232 serial ports at rates as high as 115,200 baud. The library supports the Hayes Smartmodem command set and handles Xmodem and Kermit data-transfer protocols. It works with C++ compilers from Borland and Zortech.

Greenleaf Software, 16479 Dallas Parkway, Suite 570, Dallas, TX 75248, (800) 523-9830 or (214) 248-2561.

Circle **218** on Reader Service Card

A QUARTET FROM GENUS

Genus Microprogramming has released four programmers' tools designed to provide enhanced graphics facilities for DOS programmers.

GX Effects ($199, or $399 with source) provides special effects, including fade, wipe, roll, random, slide, spiral, and explode. The package offers animation and sprite capabilities, more than 50 dissolve effects, and palette cycling and fading. The library includes a music-definition language and a Sound-Blaster voice-file interface.

GX Games ($59) is a set of five ready-to-run, executable games (BlackJack, Puzzler, Apples & Oranges, MindPlus, and SpaceMiner) with complete source code in C, Pascal, and BASIC. The package illustrates the use of Genus's line of GX development tools.

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

FOLLOW THE LEADERS IN SOFTWARE DEVELOPMENT

"I've been at every Software Development Conference held. SD is the event where the PC software development industry happens. It's the single conference that gives the software tools community–users, developers, and commentators – a sense of identity.

The SD conferences are a celebration of the culture of software development. For a few days, a few times a year, all the genuinely great programmers from around the world come together, to meet with each other and with the only people who understand the significance of their efforts and achievements. Overall, I'd have to say that my experiences at the various Software Development Conferences have been very important in shaping my development as a programmer."

Jim McCarthy
Vice President, Research & Development
Whitewater

February 25-28
Santa Clara Convention Center
Santa Clara, CA

SOFTWARE DEVELOPMENT '92

By programmers for programmers

Since its inception in 1987, Software Development has grown in size, scope, and importance. Why? Just ask Jim McCarthy, Larry Constantine, or any one of the industry leaders who make Software Development a "priority event" every February. Simply put, missing Software Development could mean missing the solution to your toughest programming problem.

Here's a sampling of the exciting new tracks in the works for 1992.

- **Nuts and Bolts** – highlights new strategies and techniques for assembly, compiler optimization, networkable applications, algorithm design, hardware manipulation, and more

- **Interoperability** – insight into distributed databases, downsizing, client-server, developing networkable applications, cooperative processing, X programming, and more

- **Testing, Debugging, and Maintenance** – ins and outs of reverse engineering, profiling, online testing, performance analysis, and documentation

- **Programming Futures** – up-to-the-second information from the sources on what's happening with pen-based computing, end-user programming, parallelism, Apple's PINK, and the Patriot Partners

Plus, classes from the best tracks of the 1991 conference: C/C++, Windows programming, analysis and design, object-oriented programming, OS/2, and user interface design

Tap into our information pipeline today!

Just send in the attached information card and we'll keep you posted on advance registration savings and up-to-the-minute news on lectures, workshops, and the largest programming products exhibition of its kind.

All this will take place **February 25-28, 1992** at the Santa Clara Convention Center. So get on our mailing list today and get ready for "The Premier Event" for Software Developers.

Circle 51 on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

The GX Text library ($149, or $299 with source) allows programmers to display bitmapped text in graphics mode. The library includes more than a megabyte of fonts and works with any typestyle in the GEM format. Using GX Text, programmers can manipulate foreground and background colors, transparent text, justification, bold, underscore, and fixed and proportional spacing. Library functions can be called by C, Pascal, BASIC, Fortran, assembly language, and Clipper programs.

The Proteus 5.0 prototyping tool ($249) lets programmers test design ideas and sample interfaces before writing code. The environment supports special video effects and, in slide-show mode, allows the creation of demo versions for applications. Proteus 5.0 works with MDA, Hercules, CGA, MCGA, EGA, VGA, and Super VGA video adapters.

Genus Microprogramming, 2900 Wilcrest, Suite 145, Houston, TX 77042, (800) 227-0918 or (713) 870-0737.

Circle **219** on Reader Service Card

KNOWLEDGESHOP

KnowledgeShop ($295), from **Decision Software,** allows programmers to build, test, and embed knowledge-based decision-making capabilities into C and C++ programs. The package—which employs mathematical, heuristic, empirical, and other reasoning techniques—includes tools for testing and debugging decision modules. Once the programmer is satisfied that the AI module is complete, KnowledgeShop can generate C code suitable for compilation with any ANSI C compiler.

Decision Software Systems, 160 West Street, Cromwell, CT 06416, (203) 632-7570.

Circle **220** on Reader Service Card

WINDOWS INTERFACE TOOL

Borland's Resource Workshop ($49.95) helps programmers create the interface portions of Windows programs written in C++, Pascal, and other languages. The software tool helps programmers to create or customize Windows resources such as icons, dialogs, fonts, and bitmap graphics. It includes 64 graphical icons.

Borland International, 1800 Green Hills Road, Scotts Valley, CA 95067, (800) 331-0877.

Circle **221** on Reader Service Card

HALO FILE FORMAT LIBRARY

The Halo Image File Format Library (HIFFL), available from **Media Cybernetics** for $249, allows C programmers to read and write bitmapped image files in formats such as TIFF, PCX, BMP, and the HALO CUT format. The Windows version of HIFFL sells for $349. Both libraries provide data compression capabilities, including LZW, PackBits, RLE, and CCIT 1D, G3, and G4. HIFFL is distributed as an object library for DOS and as a DLL under Windows.

Media Cybernetics, 8484 Georgia Avenue, Silver Spring, MD 20910, (301) 495-3305.

Circle **222** on Reader Service Card

OBJECT COMM 1.50

Object Comm 1.50 ($75, including source), from **Sycamore Microsystems,** is an OOP communications source code library for Turbo Pascal 5.5 and 6.0. It provides extensible, reusable OOP classes for communications programming projects. All communications libraries include serial-port drivers, file transfers, and terminal emulation.

Sycamore Microsystems, 614 Lynn Drive, Berrien Springs, MI 49103, (616) 471-5571.

Circle **223** on Reader Service Card

WINDOWS TOOLS

Rational Systems has introduced three tools for Windows programmers. BigWin, a 32-bit Windows application extender, allows applications to be developed and run in Windows 3 enhanced mode. Using the product, programmers can convert 16-bit applications to 32 bits by modifying and recompiling a **make** file. BigWin applications can be linked with 16- or 32-bit DLLs and can be debugged using the company's Instant-D debugger.

WinServe streamlines Windows development by providing a way to run existing programs under Windows 3. Programmers need only rewrite the application's front end; the back end continues to run under DOS, and WinServe provides communications between the two parts of the program.

DOS/4G allows DOS-extended applications to run as DOS programs under DPMI-compliant operating systems such as Windows 3.0 or 3.1. Users can run 32-bit applications in protected mode. Because the product's addressing model provides a linear rather than a segmented address, extended memory is easier to access. The product is also compatible with VCPI and XMS extended-memory standards and supports TSRs.

Prices for each of the products begin at $5,000.

Rational Systems, Inc., 220 North Main Street, Natick, MA 01760, (508) 653-6006.

Circle **224** on Reader Service Card

QUICKC FOR WINDOWS

Microsoft has introduced QuickC for Windows, a Windows-hosted development environment and compiler for C programmers. The product allows C programmers to write Windows applications without recourse to the Windows 3.*x* SDK, according to the company.

QuickC for Windows includes QuickCASE:W, a screen designer that generates commented C code for windows, menus, and other elements of the user interface. Also provided are an editor, a Windows-hosted debugger, and the QuickWin Library, which lets programmers port DOS C programs to Windows simply by relinking.

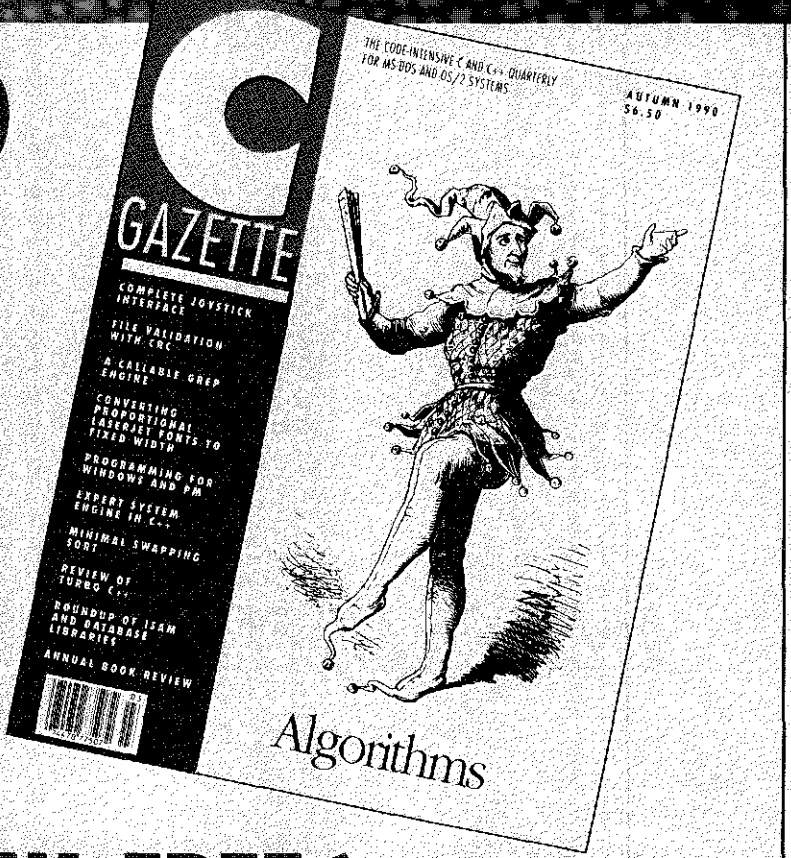
QuickC for Windows is upward-compatible with Microsoft C 6.0 and the Windows 3.*x* SDK. It retails for $199, or for $89.95 to registered users of Microsoft C compilers. Add-on tools, including the Windows Help compiler, CodeView for Windows, SDK debugging tools, the debugging version of Windows, and Charles Petzold's *Programming Windows, 2nd edition,* can be purchased separately, or cost $150.95 if they are bought as a set. A package including the Help compiler, CodeView for Windows, and SDK debugging tools retails for $49.95.

Microsoft's COBOL professional development system version 4.5 ($900) is designed to ease COBOL programming for the Windows environment. Version

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

INTENSIVE. IN-DEPTH.

PACKED WITH C SOURCE CODE.



(& RIGHT NOW, FREE.)

DATA COMPRESSION • HASH TABLES • DOS INTERFACE • MICE • LASER PRINTERS • DEBUGGING • DATA EDITING ALGORITHMS • FONTS • GRAPHICS • TREES • BINARY SEARCH • TELECOMMUNICATIONS • AND MORE!

C Gazette is the only code-intensive C and C++ publication. Each issue offers cover-to-cover code with supporting text. Topics are treated in depth and presented in a clear, cogent manner that emphasizes real-world solutions.

For a **FREE preview copy,** call **1-800-234-0386.** We'll send you the next issue of ***C Gazette*** and invoice you for a one-year subscription. If you are not satisfied, write cancel on the invoice. The free issue is yours to keep.

1 year (6 issues) **$24.95**

49110

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

4.5 includes the QuickWin library, which allows programmers to port standard COBOL applications to Windows without modifying code. The product supports development of Windows DLLs and of Windows-based programs written in COBOL.

Microsoft Corporation, One Microsoft Way, Redmond, WA 98052, (206) 882-8080 or (800) 426-9400.

Circle **225** on Reader Service Card

32-BIT WINDOWS ADK

MetaWare's 32-bit application development kit ($495) can be used with the company's High C compiler to produce 32-bit applications for Windows. The Windows ADK also supports the company's 32-bit runtime library, and applications developed for 16-bit Windows can be ported with minimal changes. The ADK includes a Windows Supervisor, which facilitates the loading of 32-bit code and communication between 16-bit Windows and the 32-bit application. Also included is a 32-bit source-level debugger.

MetaWare Inc., 2161 Delaware Avenue, Santa Cruz, CA 95060-5706, (408) 429-6382.

Circle **226** on Reader Service Card

ACTOR 4.0

The Whitewater Group has released version 4.0 of Actor, its application development platform for Microsoft Windows. Actor Version 4.0 includes an object-oriented language, development environment, and libraries, providing all the tools needed to write Windows programs. Enhancements include database-independent SQL libraries and DDLs for Paradox, Excel, SQL Server, dBASE, DB2, Oracle, and OS/2 EE.

Actor 4.0 comes in two versions. The $249 base package includes the language and development environment. Actor 4.0 Professional ($495) adds database access libraries, the Whitewater ObjectGraphics Library, and the Whitewater Resource Toolkit.

The Whitewater Group, 1800 Ridge Avenue, Evanston, IL 60201, (800) 869-1144 or (708) 328-3800.

Circle **227** on Reader Service Card

DOS EXTENDER ENHANCEMENT

Phar Lap has added support for Borland C++ and Microsoft Fortran to version 2.0 of the 286|DOS-Extender SDK ($495). The SDK can now be used to develop extended-DOS applications in C++, Fortran, or Microsoft C. The product is compatible with the Borland and Microsoft toolkits, including Borland's Turbo Debugger and Microsoft's CodeView. It allows 16-bit protected-mode applications to access as much as 16MB of extended memory.

Phar Lap Software, Inc., 60 Aberdeen Avenue, Cambridge, MA 02138, (617) 876-1510.

Circle **228** on Reader Service Card

Fax/facts
COPIA INTERNATIONAL

Introducing the Fax-on-Demand System That's Above and Beyond The Call.

Immediately respond to your customer's needs with FaxFacts® Fax-on-Demand. Be there with on-the-spot tech support, late breaking news, and sales and marketing information without adding another member to your staff!

- Inexpensive one-call and full-featured call-back delivery methods.
- Utilize images & messages on line-by-line basis for Service Bureau.
- Switch between one-call and call-back configurations dynamically.
- Auto charge feature verifies credit card, debits account, issues receipt.
- Password protection for system entry & images.
- Additional features include Broadcast, DID, up to 4128 lines.

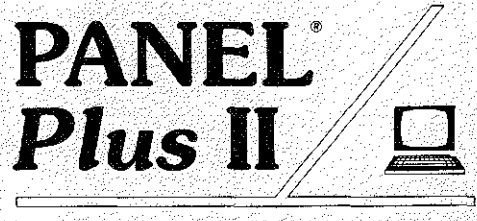
TRY THIS DEMO: 708/924-7465 DOC. NO. 889874

Copia International, Ltd., Wheaton, IL 60187 Voice: 708/682-8898 FAX: 708/665-9841

Dealer Inquiries Welcome

Contact this advertiser directly

PANEL® Plus II



Screen Generator and User Interface Library for 'C'

PANEL *Plus* II *features:* Interactive screen design editor • C and MS-FORTRAN code generators • library of user-interface functions • pop-up fields/windows • multi-line fields • mouse support • scrolling regions/scroll bars • help boxes • pull-down menus • custom validation • shadow borders • complete **library source**, with low-level source and headers for all supported systems (DOS, OS/2, Unix, VMS) • interfaces to popular PC **graphics libraries** • OS/2 DLL • **no royalties.**

Also available: **Utility Source Option** which allows **PANEL *Plus* II** to be ported to any suitable development platform. *New in v2.1:* **CUA/SAA** key assignments and edit functions, CUA 'function key area' support, new menu styles, word-wrap editing, QH on-line help, DOS-extender interfaces, extensive example programs.

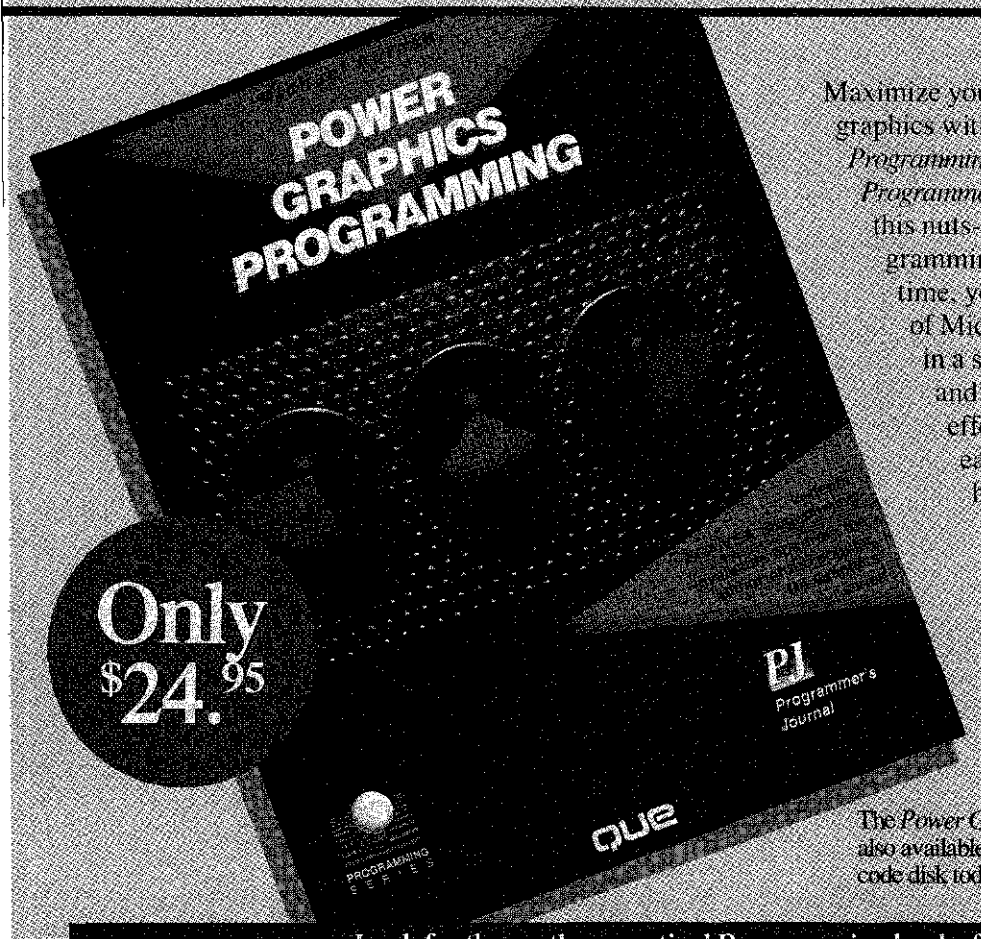
Panel *Plus* II license fee per programmer: **$495** (including library source and 700-page manual). **Panel *Plus* II** with utility source: $1195. Call for free demo disk. Instant info via fax: dial (708) 924 7465, press 3709#

Roundhill Computer Systems Limited
1342 Avalon Court, Wheaton IL 60187
Telephone: **(708) 690 3737** • Fax: (708) 665 9841 • BIX: *join roundhill*
UK Head Office: POB 14 Marlborough, Wiltshire SN8 1LG • Tel: **0672 84535** • Fax: 0672 84525

Contact this advertiser directly

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Graphic Results!



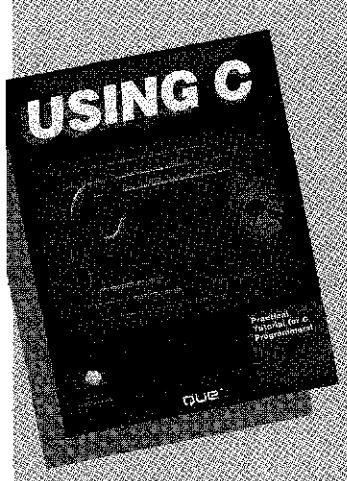
Only $24.95

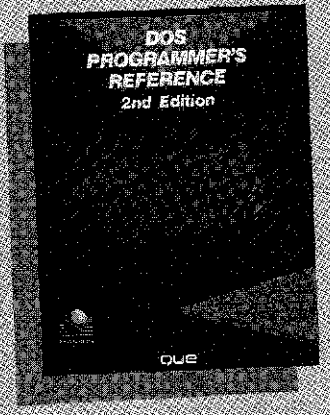
Maximize your EGA/VGA programming graphics with Que's *Power Graphics Programming*! Que has teamed up with *Programmer's Journal* to bring you this nuts-and-bolts graphics programming guide. Now, for the first time, you can have the expertise of Michael Abrash consolidated in a single volume. His insights and approach to efficient, effective programming have earned him high acclaim for his "On Graphics" series. If you are seeking that elusive graphic edge, you can't be without *Power Graphics Programming*!

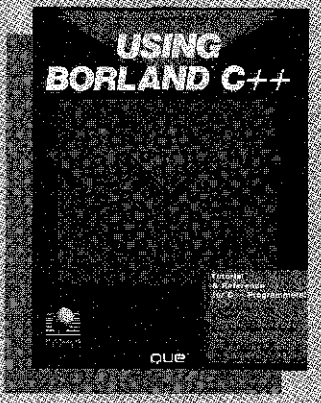
Power Graphics Programming is only available from *Programmer's Journal*. Call 800-234-0386.

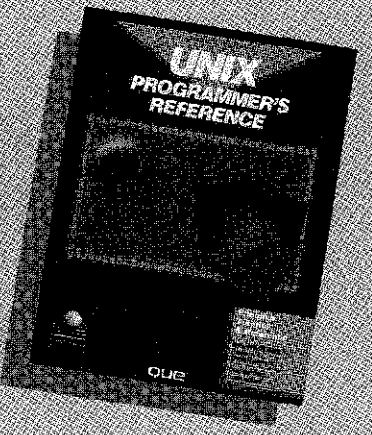
The *Power Graphics Programming* code disk is also available for $14.95. Order your book and code disk today!

Look for these other practical Programming books from Que









Available in better bookstores and computer stores everywhere.
To order, call 800-428-5331, ext. PJ01.

que®

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

FIGHT PIRACY & PROTECT YOUR PROGRAM $$$'s!

Since 1986, companies worldwide have been choosing Az-Tech security products. If you demand the strongest protection available, why not choose one of these "proven leaders":

- **EVERLOCK** Copy Protection
- **EVERTRAK** Software Security
- **EVERKEY II** "The Lock"

For IBM and Compatibles.
30 day money back guarantee.
Free info and demo disk available.

Az-Tech Software, Inc.
201 East Franklin, Suite 11
Richmond, MO 64085
(800) 227-0644 (816) 776-2700
FAX (816) 776-8398

Circle 110 on Reader Service Card

OPT-TECH SORT/MERGE

Extremely fast Sort / Merge / Select utility. Run as an MS-DOS command or CALL as a subroutine.
Supports most languages and filetypes including Btrieve and dBase. Unlimited filesizes, multiple keys and much more!

MS-DOS, Windows **$149.**
OS/2, UNIX **$249.**

Opt-Tech Data Processing
P. O. Box 678
Zephyr Cove, NV 89448
(702) 588-3737

Circle 111 on Reader Service Card

EZ-DISKLONE ONLY $29.95 +S&H

•Picks up where DOS's DISKCOPY leaves off! •Read source diskette only once. •Quickly duplicates target disks in ONE PASS (360K, 720K, 1.2M, 1.44M) on your own PC/XT/AT/etc. © •Command line or Menu operation •DOS verifies •Even faster with pre-formatted disks. •Ask about EZ-DisKlone PLUS ($79+S&H) with disk imaging/HD storage, compare, fastformat, EMS/XMS support for speed, etc.

EZX 17077 Texas, Box 58177, Webster, TX 77598
Discover, VISA, MC, AX Accepted
CC Orders: 1.800.800.2468
InfoVoice: 713.280•9900 FAX•0525

Circle 112 on Reader Service Card

New Version 2 - Execute up to 256 C Functions Concurrently in One Program with Conquerrent C

- **Great for real-time, data acquisition, clock displays, background printing, simulation, process control, games.**
- **Transparent reentrancy protection for most large model library functions.**
- **Flexible stack location and size management for each task.**
- **All memory models supported in Turbo and Microsoft C on DOS 80x86 PC's.**
- **New Low Price - $45 complete, no royalties, 30 day satisfaction guarantee.**

Available from:
Interstitial Software Systems
321 E. Buckingham Rd., Ste. 140
Garland, Texas 75040

Circle 113 on Reader Service Card

PL / 1 on DOS?

- ANSI Standard
- Native Mode Compiler
  Off load your mainframe development to PC's

DISCUS Distribution
408-655-4505

Circle 114 on Reader Service Card

TLIB™ is FASTEST!

Times are to update a 45K library on a PC/XT. PVCS and TLIB 3.0 are from Sept '87 *PC Tech Journal*. MKS RCS 4.2 and TLIB 5.0 are newer.

TLIB™ is *BEST!*

"Do not be fooled by the fact that this is the least expensive of the five packages reviewed here - TLIB has features and power to spare" John Rex, **Computer Language**
"TLIB is a great system" J. Vallino, **PC Tech J**

- **Full-Featured Version Control for Software Professionals.** Check-in/out locking. Branching. Keywords. Wildcard and list-of-file support. Can merge parallel changes and undo intermediate revisions. Network and WORM support. **Mainframe compatible** deltas for Pansophic, ADR, IBM, etc.. Integrates with Opus™ MAKE & Slick™ MAKE.

MS-DOS $139, OS/2 $195 + shipping. Visa/MC. 5 station LAN license $419 (OS/2 $595); call for other sizes

BURTON SYSTEMS SOFTWARE
PO Box 4156, Cary, NC 27519 (919) 233-8126

Circle 115 on Reader Service Card

Why you want BATCOM!

BATCOM is a batch file compiler that compiles your ".bat" files to ".exe" files to make them faster, more professional, and more capable. **BATCOM** extends DOS with new commands so you can read keyboard input, perform arithmetic, use subroutines, and much more. In addition, **BATCOM** protects your source code, and you can distribute your compiled programs without royalties. Only **$59.95**

ORDER TODAY!

Wenham Software Co.
5 Burley St.
Wenham, Ma. 01984
(508) 774-7036

Circle 116 on Reader Service Card

BGI Printer Drivers

The BGI Printer Driver Toolkit provides BGI drivers for Epson 9 & 24 pin, HP LaserJet II, and HP PaintJet printers. Use the BGI graphics interface to effortlessly generate high resolution hardcopy with Turbo C, TC++, and Turbo Pascal. **Full driver source code included.** $89.95

PC Timer Tools

60 functions implement microsecond resolution timing, precision delays, interrupt profiling, and timer tick management. Ideal for profiling and data acquisition/process control. Supports TC, TC++, MSC, TP. **Full library source code included.** $49.95

Prices postpaid USA, elsewhere add $4
VISA & MasterCard accepted.
30 day money back guarantee

RYLE DESIGN
Purveyors of Big Science
PO Box 22, Mt. Pleasant MI 48804
(517) 773-0587 CIS:73047,1765

Circle 117 on Reader Service Card

FORTRAN

FORTRAN Toolkits
Comprehensive libraries of over 300 graphics, text, and systems service subroutines from ellipses to child-processes to I/O verification. High-speed video drivers, from CGA to Super VGA, are all internal. Option rich, easy to use. No royalties. 300+ pg. manual, 8,000 lines of sample source. **Price $215.**

DOCUMENT'er and DIAGRAM'er
Makes source code understandable. Indexes, cross-references, and diagrams the structure of source code. No changes to source required. Nested includes, recursion, and annotations handled automatically. **Price $165.**

IMPULSE *Engineering*
PO Box 190206,
San Francisco, CA 94119-0206
(415) 788-4611

Circle 118 on Reader Service Card

THE CD-ROM DEVELOPER'S LAB™

Comprehensive CD-ROM multimedia production resource for PC and Apple developers. Proven design, management, data prep, programming, premastering and manufacturing techniques. Important specs from leading companies. Demos of off-the-shelf tools for imaging, audio and animation. Sample applications demonstrate the power of our source tools in C, Turbo Pascal.
PC or MAC @ **$395.00;**
Transportable version @ **$425.00**
Visa/MC accepted.

SOFTWARE MART, INC.
3933 Spicewood Springs Rd., Suite E-100
Austin, Texas 78759
(512) 346-7887 FAX: (512) 346-1393

Circle 119 on Reader Service Card

CONSULTING JOBS
IBM, MCI, Magnavox, etc.

Beginners/Experts Needed !

➤ **Programmer: $30 - $75/hr**
➤ **S/W Tester: $29 - $52/hr**
➤ **Tech Writer: $20 - $41/hr**

Many More Professions !

Database & Book Teach How.

Database Program, 485 p Book.

Six Different Resumes: $69.95

Authors Consultants for 18 Yrs

Madden Associates. Dept. PJ
2039-F Civic Center Drive
N. Las Vegas, NV 89030

Circle 120 on Reader Service Card

INTEL® 80486 MACRO DISASSEMBLER

MD86 (Masterful Disassembler) is the most comprehensive **macro** disassembler on the market. Interactive by design not after-thought. Features include:

- ► User defined macros with arguments
- ► Auto code/data separation
- ► Auto commenting and label naming
- ► Command menus and help screens
- ► Customizable disassembly options
- ► Comprehensive, indexed, manuals
- ► Fully supports 8086/87 thru 80486/87
- ► COM/EXE/SYS/memory/other
- ► Cross reference and more!

Still only $67.50 + tax ($1.50 s/h)

C.C. SOFTWARE
1907 ALVARADO AVE.
WALNUT CREEK, CA 94596
(510)939-8153

Contact this advertiser directly.

Get the attention your product deserves.

Placing an ad in **PJ BILLBOARD** is an easy, low-cost way to reach potential buyers. Display ads are 1⅝" wide by 2⅜" high. For more information or to reserve a space, call our ad department at **(800) 234-0386.**

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

PJ BILLBOARD

LOOK

at the savings in development time you can get with DOCZ to automate subroutine and program documentation. Discover how to use DOCZ to build an object-oriented development environment to share and re-use your software modules. FREE MSDOS demo disk and samples. 60-day money-back guarantee. VISA/MC accepted. $195/ Single-User. Start benefiting NOW!

DOCZ™

Do It Better With Toolz!

Software Toolz, Inc.
8030 Pooles Mill Drive
Ball Ground, GA 30107-9610
(404) 889-8264
Voice-mail: (800) 869-3878

Circle 121 on Reader Service Card

The Upper Deck Editor
A Serious Editor For Windows 3.0

Real UNDO (300 levels), regular expressions, search across multiple files, compiler support, split-screen file comparison, fully customizable.

INTRODUCTORY PRICE $75.

Call for our fully functional
DEMO DISK!
(619) 741-1075

Upper Deck Systems
P.O. Box 2751, Escondido, CA 92033

Circle 122 on Reader Service Card

DataLIB™ Windows & DOS

Attention Windows and DOS software developers: DataLIB is the quickest way to add professional, fast, and flexible data file import/export capabilities to your application. Read and write most major spreadsheet, database and ASCII file formats including Lotus, Excel, Symphony, DBASE, Quattro, DIF, SYLK, CSV, TAB and TXT files. DataLIB includes full support for major C compilers (MSC, BC++) and Windows development systems (Visual Basic, C, C++, Actor, Pascal). Great documentation and support! **Reduce development time!! NO ROYALTIES!!**

DatTel Communication Systems
3508 Market Street, Suite 415, Phila, PA 19104
Call: 215-564-5577 Fax: 215-243-3705

Circle 123 on Reader Service Card

'Quick and Clean' COBOL®

The COBOL Report Activator *(COBRA)*:

- Generates efficient, structured COBOL programs
- Is easy to learn and use
- Accepts your COBOL code and copybook members
- Produces audit trail reports
- Uses one or two file input
- Allows site defined defaults
- Available in mainframe & PC

Olivadoti Associates
(Formerly *COBRA* Systems)
P.O. Box 750817
Petaluma, CA 94975-0817
(707)763-5123

Circle 124 on Reader Service Card

DEBUG WITHOUT SOURCE

Quaid Analyzer debugs by showing interrupts as they occur. You can even watch DOS boot load without hardware changes. The manual tells how to deal with programs that don't want to be traced - we use it on copy protection. You scroll through memory like text in an editor, and never type a command. US $200.00

Quaid Software Ltd Dept PJ
45 Charles St East Third Floor
Toronto Ontario Canada M4Y 1S2
(416) 961-8243
FAX: (416) 961-6448

Circle 125 on Reader Service Card

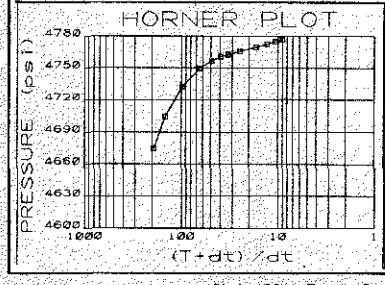
INGRAF 5.00

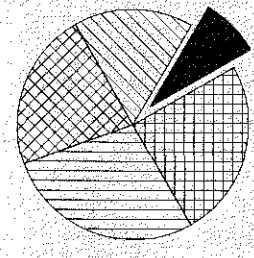
A Multi-Device Graphics Library for Scientific, Engineering and Business applications. Supports Video(CGA, EGA, VGA & Hercules), Printers (EPSON, Okidata, HP LaserJet, HP Paintjet & Postscript) and Pen Plotters(HPGL and DM/PL).

C, FORTRAN-77, PASCAL, MS Professional BASIC and QuickBASIC bindings.

FULL SOURCE CODE • NO ROYALTIES • $350.00

Sutrasoft
10506 Permian Dr.
P.O. Box 1733
Sugar Land
TX 77487-1733
U.S.A.
Tel: 713-491-2088
Fax: 713-240-6883





Circle 126 on Reader Service Card

PJ CODE IS AVAILABLE!

All the source code from the articles in this issue is available on disk from

Programmer's Journal
P.O. Box 70167
Eugene, OR 97401-0110
800-234-0386
503-747-0800

Code disks are $10.00 each. Please see page 77 for ordering information.

PJ Code Diskscriptions are available for $39.95. Look on the card opposite page 12 to order.

The code listings are also available for downloading from several Bulletin Board Systems around the country. See page 60 for listings.

Contact this advertiser directly.

Useable User Guides!

Your customers deserve user guides they can understand. For writing samples and a free estimate, call:

Extra Mile Writing Services
(619) 672-3388
11057 Carlota Street
San Diego, CA 92129

Circle 127 on Reader Service Card

FORTRAN TO STRUCTURED C / FORTRAN
TRANSLATOR

FORTRAN PROGRAMMER'S ASSISTANT II converts spaghetti FORTRAN into neat, maintainable C code. Run-time library source and C tool set included. It will make your conversion process easy and will significantly reduce your conversion and future maintenance costs. ASSISTANT II Plus also produces structured FORTRAN. Easy to use. At low prices. A must re-engineering tool for FORTRAN programmers.

MICROTOOLS
P.O. Box 2745 PJ
Santa Clara, CA 95055-2745
(408) 296-3478
FAX (408) 243-7688

Circle 128 on Reader Service Card

POWER GRAPHICS PROGRAMMING

If you do EGA/VGA programming, then *Power Graphics Programming* is a "must have" for your reference library. This Que book is a compilation of Michael Abrash's first four EGA and nine VGA "On Graphics" articles published in *Programmer's Journal*. It contains great theory, invaluable tips, and nuts-and-bolts ASM and **C code** listings. A companion code disk is available. Order your book and disk today.

SPECIAL

P.G.P. Book & Code Disk ..... $29.90 *
(Regular Price: Book $24.95, Disk $14.95)
* Plus $1.50 handling charge per order. Free shipping anywhere in the U.S. Additional shipping charges to Canada, Mexico, and other foreign destinations.

Order your book and disk direct from:
Oakley Publishing
Call toll free with your VISA/MC
800-234-0386
Outside the U.S., call 503-747-0800

Contact this advertiser directly

Coming Next Issue:
10TH ANNIVERSARY & GRAPHICS

January/February 1992 Vol. 10.1
Advertising space reservations are due:
Nov. 1, 1991
Materials are due:
Nov. 8, 1991

Please call 503-747-0800

Ask for Jay or Maryann in the advertising department to reserve your space.

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

How much is that multiprocessor?

By Hal W. Hardenbergh

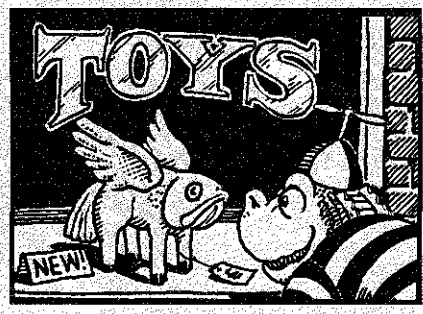


ILLUSTRATION BY JIM CARPENTER

Right after World War II, many believed that the average upper-class American family would have a car, a boat, and an airplane. Each has an internal combustion engine, fuel tank, and steering mechanism. Hmm. Prototypes were actually built of a vehicle that was fish, fowl, and horse. The wings detached for road or aquatic use. Brilliant idea.

For the past 16 years lots of people have wanted to build powerful supercomputers using massive arrays of microprocessors. How many of you remember the first, never-built Imsai product? A very small number of such computers actually exist today. The total installed base is about equal to the number of parallel processing companies that have gone belly up. The BBN Butterfly is the latest parallel casualty.

As a spinoff of this parallel mania, lots of folks in the desktop computer world (encompassing both DOS PCs and Unix workstations) are excited about "symmetrical multiprocessing." I'm not, and you shouldn't be, either. Explanation follows.

By 1989, all high-performance desktop computers used caches to buffer the slow DRAM memory from the fast CPU. By the end of that year all four principal desktop engines (486, SPARC, MIPS, and 88000) were shipping with 25-MHz CPUs that executed roughly one instruction per clock. The caches did such a good job of buffering the DRAM that only 20% or so of the DRAM bandwidth was actually being used. Computer designers hate to waste resources, so much thought was devoted to what could be done with that 80% of unused DRAM bandwidth.

Why not tie *four* microprocessors onto the DRAM? Each has its own cache buffer, and we can tie the four caches into the DRAM using the system bus. Sure, there'll be some bus contention, but we should get almost a fourfold performance boost, right? So we have multiprocessing (because more than one CPU services the global DRAM memory), and it's symmetrical (because the CPUs are all the same and have equal priority). This makes beautiful sense; in fact, the 88000 was designed to multiprocess from the git-go and so had built-in hooks to bus-snoop and handle the cache-coherency problem.

Having decided that symmetrical multiprocessing (SM) was wonderful, buildable, and therefore the wave of the desktop future, lots of IC companies have spent the last year or so designing integrated bus-snooping cache controllers. One such company is Intel, a highly successful company. If it's jumping on the SM bandwagon, that's surely the bandwagon of choice, right? Alas, wrong. Intel also produced the iAPX 432, the most spectacular failure in the history of integrated circuits. Nobody hits a home run every trip to the plate.

There are both hardware and software reasons why there's no SM in your desktop's future. The hardware reasons first. If a cache-buffered 25-MHz CPU that executes roughly one instruction per clock (thus excluding 386s) eats 20% of the DRAM bandwidth, then a single 50-MHz CPU will gobble 40%! Have you noticed the speeds of CPUs that are being shipped today? Do you realize the CPU speeds that are coming down the pike in the near future? There will soon be a flock of CPUs that will execute more than one instruction per clock! A 66-MHz 486 with an external 33-MHz system clock will eat about 50% of the bandwidth, and it's my rough guess that initial versions of both the 586 and the MIPS R4000 will use 65 to 75% of the bandwidth of a conventional 32-bit DRAM system. Sure, you can always build a super-expensive DRAM system—use a four-way interleaved 128-bit bus, for instance—but in a cost-sensitive PC?

Symmetrical multiprocessing is a neat idea whose time has passed. Say, wasn't there an article in the May/June 1988 *PJ* ("Memory Trends") that predicted that CPUs would soon be a lot faster than memory, and that the limiting factor of computer speed would be how much money you spent on the memory system?

You won't have an SM system on your desk, so you won't have to worry about how to program it to get that theoretical speedup (Dubois and Scheurich, "Synchronization, Coherence, and Event Ordering in Multiprocessors," *Computer,* Feb. 1988). It seems that anybody can get a 1× speedup on a uniprocessor (profound, that), but getting a fourfold speedup using four processors is a task suited to a supernatural being. A single 66-MHz 486 will almost certainly outrun a 25-MHz 486-based 4-CPU SM, even though the *theoretical* maximum performance is 33% slower. And, being a uniprocessor, it can be programmed by ordinary humans like you and me.

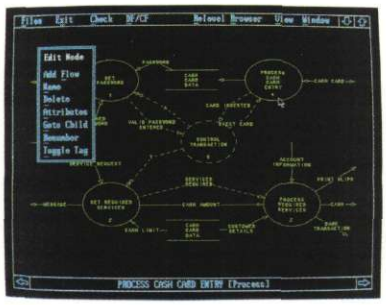
The local street price of a complete 33-MHz 486-based personal computer, less display and hard disk, is now under $1,000! I'd rather buy one of those and continue to run all that software I originally bought for my 8088 box than have an expensive symmetrical multiprocessor for which there is *no*—repeat *no*—application software.

I'm looking forward to buying an affordable 586-based PC. The 586 is almost a symmetrical multiprocessor in a single chip. It'll have more than one identical integer processor on chip and will be able to execute more than one integer instruction per clock. And it'll have hardware logic to handle instruction sequencing and data dependencies, which avoids the programming problem. My favorite thing about the 586 is that Intel has finally shed the handicaps imposed by the old IEEE 754 committee and is building a real, genuine FPU into the chip. Another religious issue bites the dust.

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Finally, an affordable CASE product that plays with the big boys.

OpenSELECT CASE teams you up with all the majors for just $495.



OpenSELECT CASE will team you with CASE big boys Cadre, IDE, and InterSolv, by sporting data transfer—based on CDIF specifications—that approaches 100% accuracy. An all star player in its own right, OpenSELECT CASE is the first PC based CASE tool that capitalizes on your current or future work-station investment.

In short, OpenSELECT CASE enables you to field a winning team for peanuts. You can spend the big bucks on a few key players, filling out the rest of your roster at $495 a seat.

A fully integrated tool set for Analysis and Design of realtime applications, OpenSELECT CASE supports multiple methodologies, including, Yourdon, Ward-Mellor, Hatley, Hood, and Gane-Sarson.

Other stats include: Multi-User ■ Multiple output formats ■ Multi-project capability ■ Multiple diagram support ■ Easy to use ■ Data dictionary ■ Consistency checking ■ Open Architecture ■ Reporting

This is no minor league product. Distributed by Meridian Software Systems, the innovator that revolutionized PC based Ada development with a product that sold 10,000 units in 6 months, the only thing minor about OpenSELECT CASE is its price.

The season is a short one. This special $495 price is only valid until November 30, 1991. **So call now to order OpenSELECT CASE 1(800)221-2522.** You can't lose: If you're not satisfied, you can return the product for a full refund within 30 days.

Hardware requirements are IBM PC or compatible, 450k of free RAM, DOS 3.0 or greater, 3 MB free disc space, and mouse (not required).

Meridian Software Systems, Inc.,
10 Pasteur Street., Irvine, CA 92718

OpenSELECT CASE is a registered trademark of Meridian Software Systems, Inc. Select CASE is a registered trademark of Select Software Tools. All product names are registered trademarks of their respective manufacturers.

open SELECT
C A S E

Circle **55** on Reader Service Card

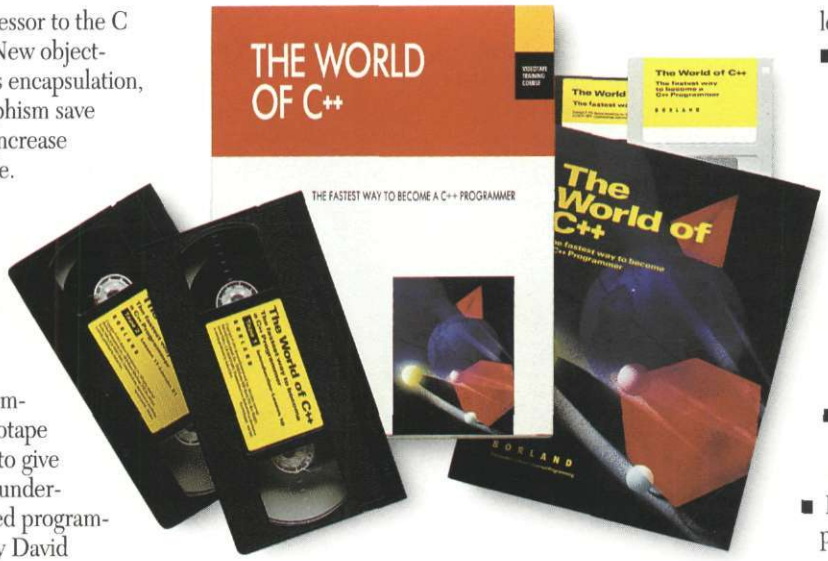
PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED

Borland International Developers Conference ▪ April 12-15, 1992 ▪ Call 1-800-942-8872

Borland just made learning C++ a lot easier

C++ is the powerful successor to the C programming language. New object-oriented concepts such as encapsulation, inheritance and polymorphism save programming time, and increase the reliability of your code.

How do you get in on the action? With Borland, of course.



The fastest path to C++ programming

The World of C++ is a complete, easy-to-follow videotape training course designed to give you a thorough, intuitive understanding of object-oriented programming and C++. Taught by David Intersimone, a noted industry expert, all aspects of the C++ language are presented in a clear logical order. And exciting animation and presentation graphics make it fun!

Master the concepts with the workbook

After each video session you're invited to pause and use the provided workbook to gain a deeper understanding of the presented concepts. Written by Bruce Eckel, an accomplished C++ teacher and voting member of the ANSI C++ committee, the workbook is a dynamic teaching tool that will capture your interest and spark creativity.

Use and modify the source code to make your own C++ programs

Complete source code is included so you can see how real C++ code is designed and constructed. And you can use this code in your own programs and your own compiler, giving you the fastest start possible in C++.

A rewarding experience

The World of C++ will have you reaping the rewards of object-oriented programming in record time. Best of all, you can forget about inconvenient classroom schedules, out-of-the-way locations, or progressing at someone else's pace. Now you can go as far as you want, whenever you want. You decide!

Just think of the new power that C++ can give you. Reusable code that can be customized and used in different applications, easier code maintenance and stronger type checking for more reliable programs, not to mention the substantially reduced cost of development. Isn't it time you learned?

Learn it all in just hours!

The World of C++ is a complete training program. In 21 easy lessons you will learn how to:

- Create new data types to represent the objects in your program.
- Hide internal data with access specifiers, and insure proper initialization and cleanup with constructors and destructors.
- Reuse code with inheritance and composition.
- Build programs with polymorphism (virtual functions) so you can easily extend them.
- Create and destroy objects at runtime as easily as you create them at compile-time.
- Use pointers and references in C++.
- Properly organize your code, and use language features such as const, inline, function overloading and type-safe linkage to make more robust programs that are easy to write *and* read.

Win a Wide-Screen TV!
That's right! Borland is giving away a Mitsubishi wide-screen TV. Call 1-800-622-1005 for more details. No purchase necessary.*

Learn C++ for only $199.95

Don't waste another minute with cumbersome old-fashioned programming techniques when *The World of C++* can be yours!

See your dealer or call now 1-800-331-0877

BORLAND

The Leader in Object-Oriented Programming

CODE: MI09

*TV sweepstakes expires December 31, 1991. Must be 18 years of age or older to enter and a U.S. or Canadian resident. Employees of Borland, its affiliates, and their families are not eligible to win. Sweepstakes is subject to all local, state and federal regulations. Void where prohibited. Winner is responsible for any taxes on prize. For a copy of the official rules, call 1-800-662-1005. Pricing is in U.S. dollars. Dealer prices may vary. Offer good in U.S. and Canada only. Copyright © 1991 Borland International, Inc. All rights reserved. BI 1417Sn

Circle **56** on Reader Service Card

PRODUCED 2005 BY UNZ.ORG
ELECTRONIC REPRODUCTION PROHIBITED